**Exercise 1: Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

* + **Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

**Solution:**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (1, 'John Doe', TO\_DATE('1950-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5.5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 7000, 6.0, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

BEGIN

FOR rec IN (

SELECT c.CustomerID, c.DOB, l.LoanID, l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

)

LOOP

IF MONTHS\_BETWEEN(SYSDATE, rec.DOB) / 12 > 60 THEN

UPDATE Loans

SET InterestRate = rec.InterestRate - 1

WHERE LoanID = rec.LoanID;

END IF;

END LOOP;

COMMIT;

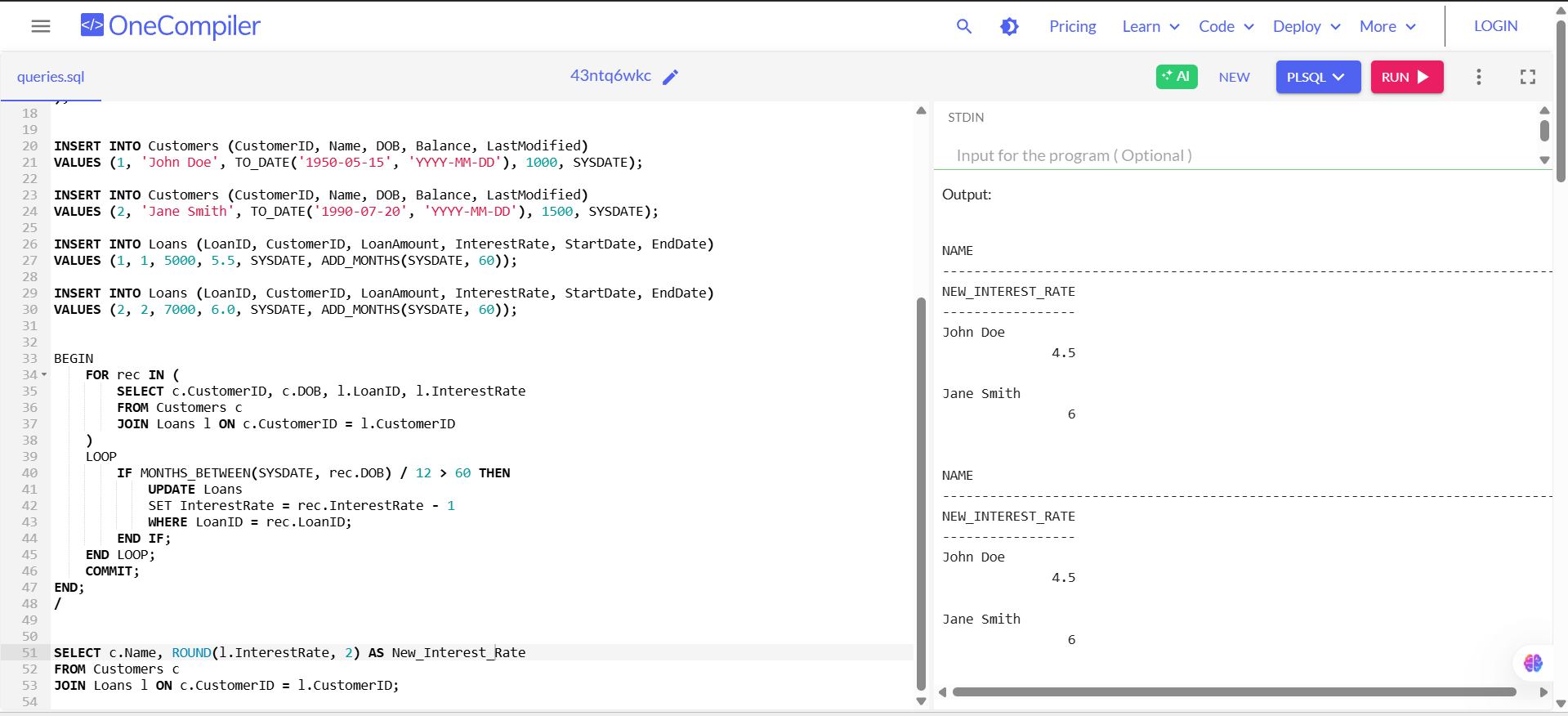
END;

/

SELECT c.Name, ROUND(l.InterestRate, 2) AS New\_Interest\_Rate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID;

**Output screenshot:**

**Scenario 2:** A customer can be promoted to VIP status based on their balance.

* + **Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

**Solution:**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE,

IsVIP VARCHAR2(5));

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified, IsVIP)

VALUES (1, 'John Doe', TO\_DATE('1950-05-15', 'YYYY-MM-DD'), 12000, SYSDATE, 'FALSE');

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified, IsVIP)

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 8000, SYSDATE, 'FALSE');

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified, IsVIP)

VALUES (3, 'Mike Lee', TO\_DATE('1982-10-10', 'YYYY-MM-DD'), 15000, SYSDATE, 'FALSE');

BEGIN

FOR exm IN (SELECT CustomerID, Balance FROM Customers) LOOP

IF exm.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = exm.CustomerID;

END IF;

END LOOP;

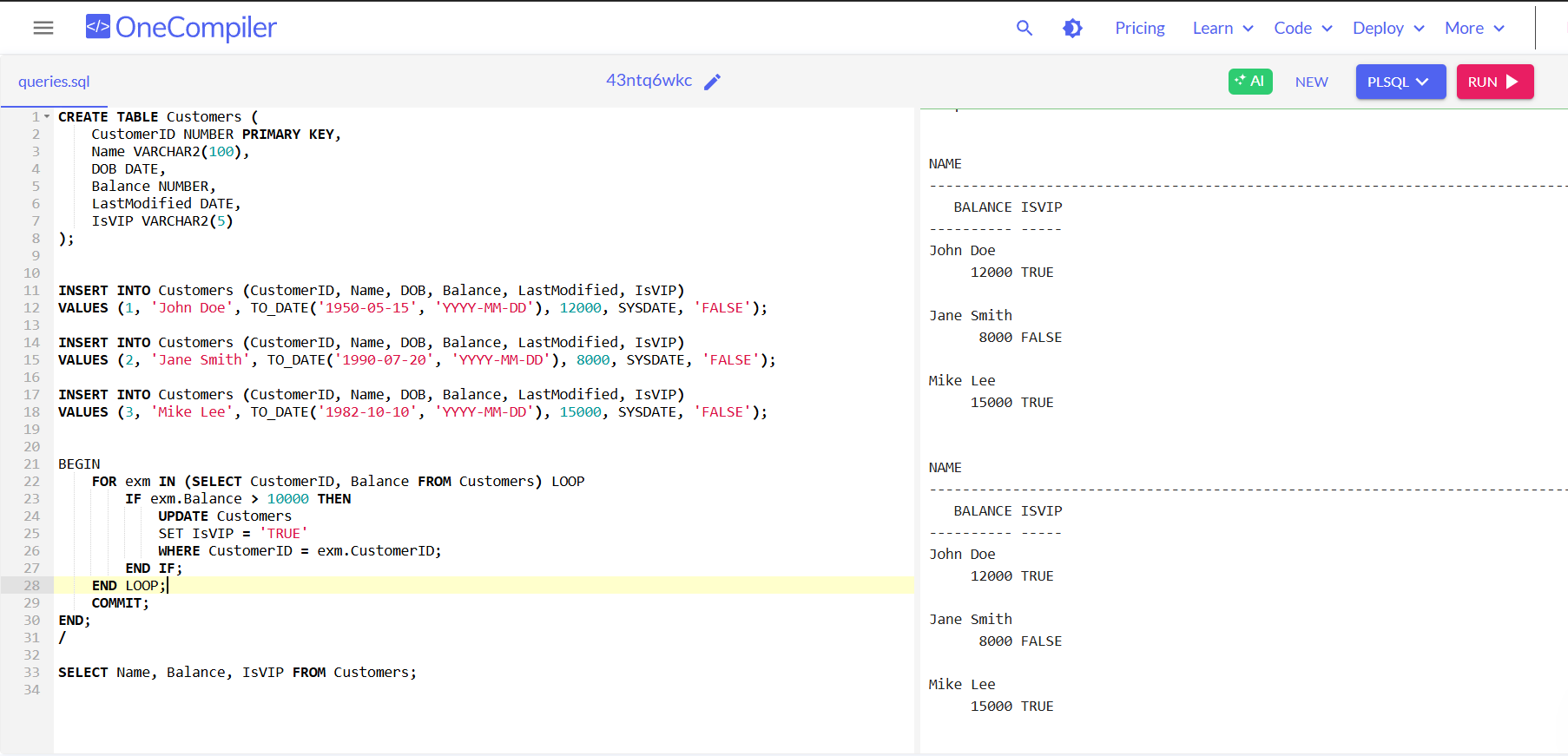
COMMIT;

END;

/

SELECT Name, Balance, IsVIP FROM Customers;

**Output :**



**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

* + **Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

**Solution:**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (1, 'John Doe', TO\_DATE('1950-05-15', 'YYYY-MM-DD'), 12000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 8000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (3, 'Mike Lee', TO\_DATE('1982-10-10', 'YYYY-MM-DD'), 15000, SYSDATE);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5.5, SYSDATE, SYSDATE + 20);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 7000, 6.0, SYSDATE, SYSDATE + 45);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (3, 3, 3000, 4.5, SYSDATE, SYSDATE + 10);

SET SERVEROUTPUT ON;

BEGIN

FOR loan\_exm IN (

SELECT l.LoanID, c.Name, l.EndDate

FROM Loans l

JOIN Customers c ON c.CustomerID = l.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

)

LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Loan ID ' || loan\_exm.LoanID ||

' for customer ' || loan\_exm.Name ||

' is due on ' || TO\_CHAR(loan\_exm.EndDate, 'YYYY-MM-DD')

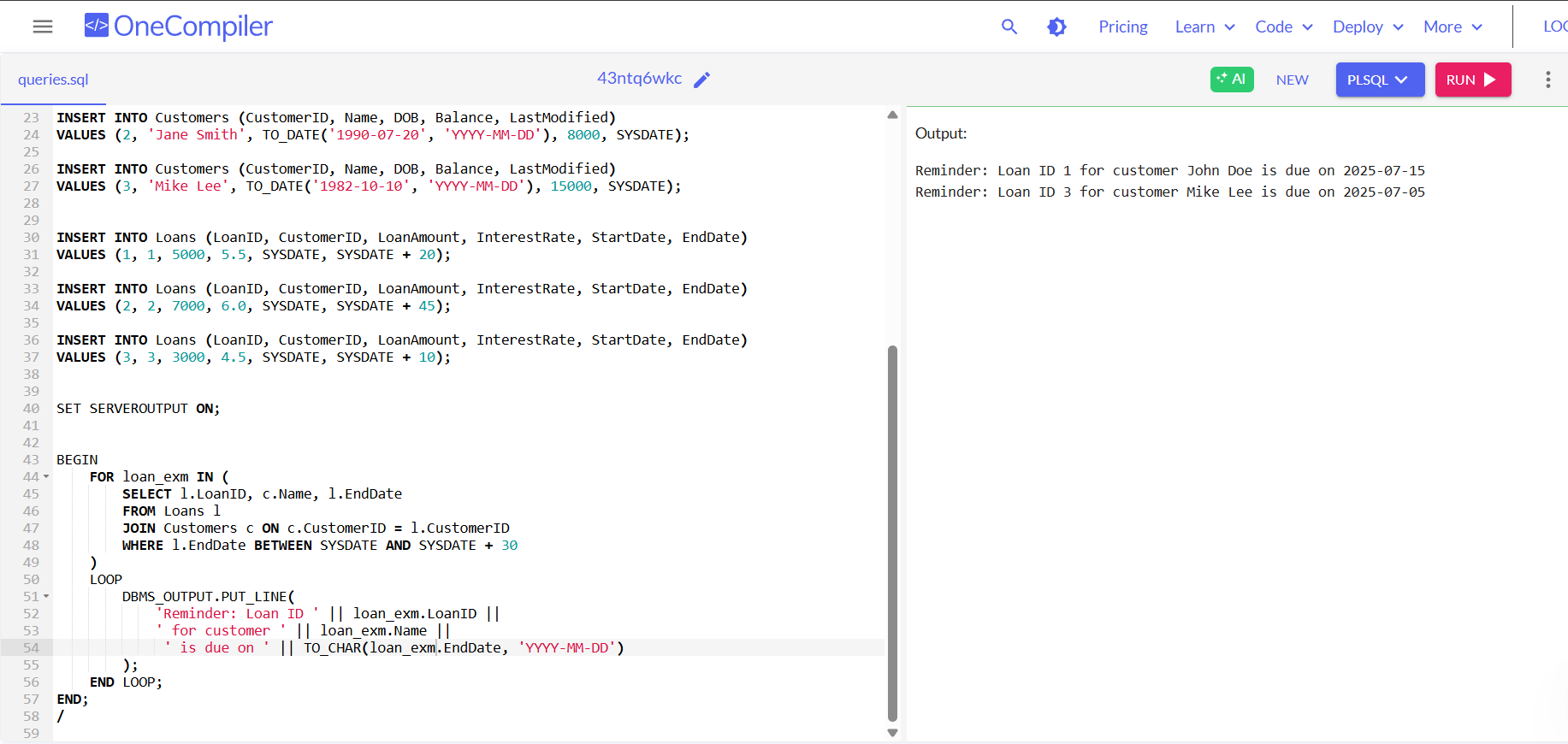
);

END LOOP;

END;

/

**Output:**

****

**Exercise 2: Error Handling**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

* + **Question:** Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

**Solution:**

SET SERVEROUTPUT ON;

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

CREATE TABLE ErrorLog (

LogID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

ErrorMessage VARCHAR2(4000),

ErrorDate DATE

);

INSERT INTO Accounts VALUES (1, 101, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 102, 'Checking', 500, SYSDATE);

COMMIT;

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_from\_balance NUMBER;

v\_error\_msg VARCHAR2(4000);

BEGIN

-- Lock and get source account balance

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account

FOR UPDATE;

-- Check if sufficient funds

IF v\_from\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in source account.');

END IF;

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account;

-- Credit to destination

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful from Account ' || p\_from\_account || ' to Account ' || p\_to\_account);

EXCEPTION

WHEN OTHERS THEN

v\_error\_msg := SQLERRM;

BEGIN

INSERT INTO ErrorLog (ErrorMessage, ErrorDate)

VALUES (v\_error\_msg, SYSDATE);

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Transfer failed. Error logged: ' || v\_error\_msg);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed and error could not be logged.');

END;

END;

/

BEGIN

SafeTransferFunds(1, 2, 200);

END;

/

BEGIN

SafeTransferFunds(1, 2, 10000);

END;

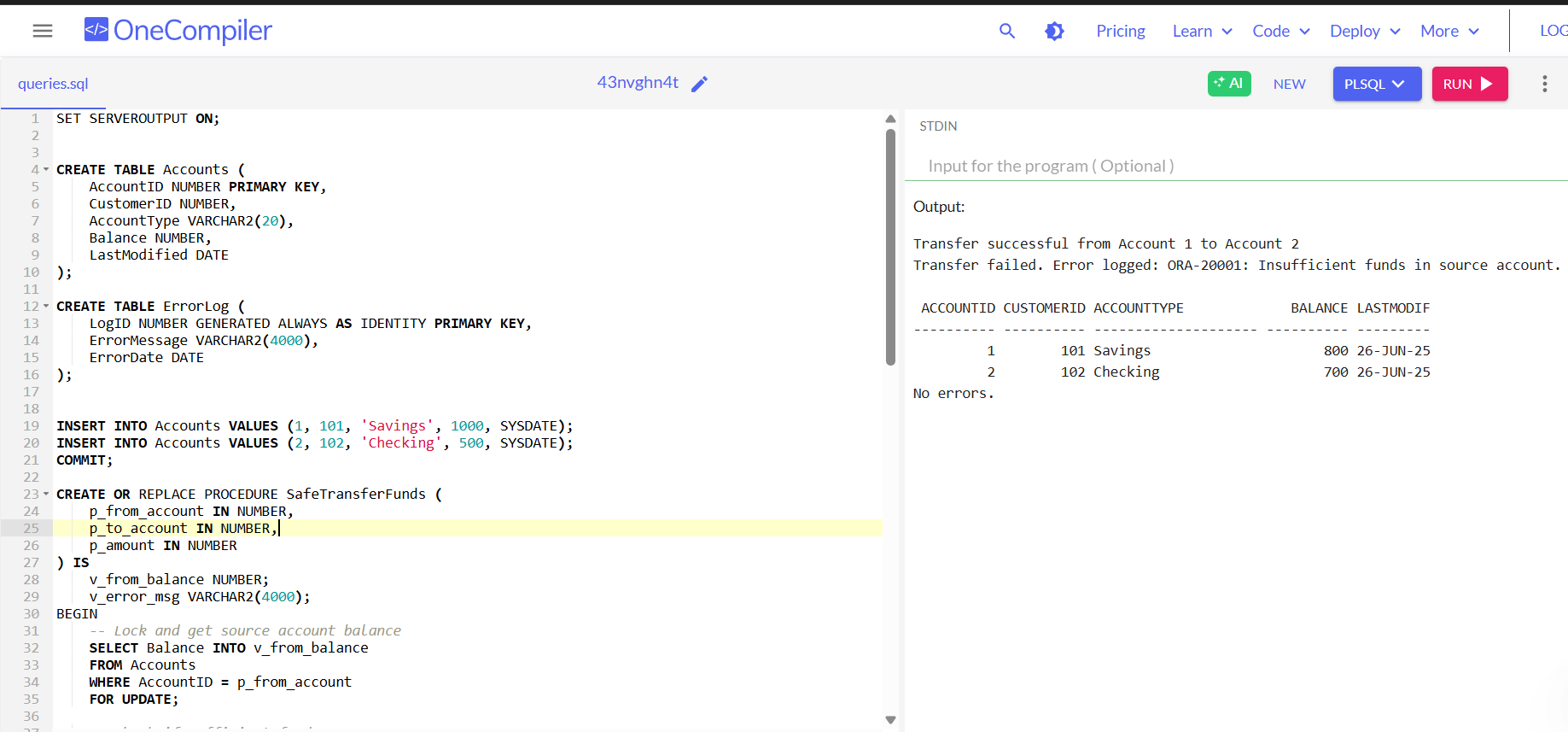
/

SELECT \* FROM Accounts;

select \* from ErrorLog;

SHOW ERRORS PROCEDURE SafeTransferFunds;

**Output:**

****

**Scenario 2:** Manage errors when updating employee salaries.

* + **Question:** Write a stored procedure **UpdateSalary** that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

**Solution:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Employees CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN

IF SQLCODE != -942 THEN RAISE; END IF;

END;

/

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES

(1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES

(2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_EmployeeID IN NUMBER,

p\_Percent IN NUMBER

) AS

v\_count NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_count FROM Employees WHERE EmployeeID = p\_EmployeeID;

IF v\_count = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Employee ID ' || p\_EmployeeID || ' does not exist.');

RETURN;

END IF;

UPDATE Employees

SET Salary = Salary + (Salary \* p\_Percent / 100)

WHERE EmployeeID = p\_EmployeeID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated for Employee ID ' || p\_EmployeeID);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error updating Employee ID ' || p\_EmployeeID || ': ' || SQLERRM);

END;

/

SET SERVEROUTPUT ON;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('==============================');

DBMS\_OUTPUT.PUT\_LINE(' UpdateSalary Testing ');

DBMS\_OUTPUT.PUT\_LINE('==============================');

UpdateSalary(1, 10); -- Existing employee

DBMS\_OUTPUT.PUT\_LINE(CHR(10)); -- vertical space

UpdateSalary(999, 20); -- Non-existent employee

DBMS\_OUTPUT.PUT\_LINE(CHR(10)); -- vertical space

DBMS\_OUTPUT.PUT\_LINE('--- Employee Salaries ---');

FOR emp IN (SELECT EmployeeID, Name, Salary FROM Employees ORDER BY EmployeeID) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Employee ID: ' || emp.EmployeeID ||

', Name: ' || emp.Name ||

', Salary: ' || TO\_CHAR(emp.Salary, '999,999.99')

);

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('==============================');

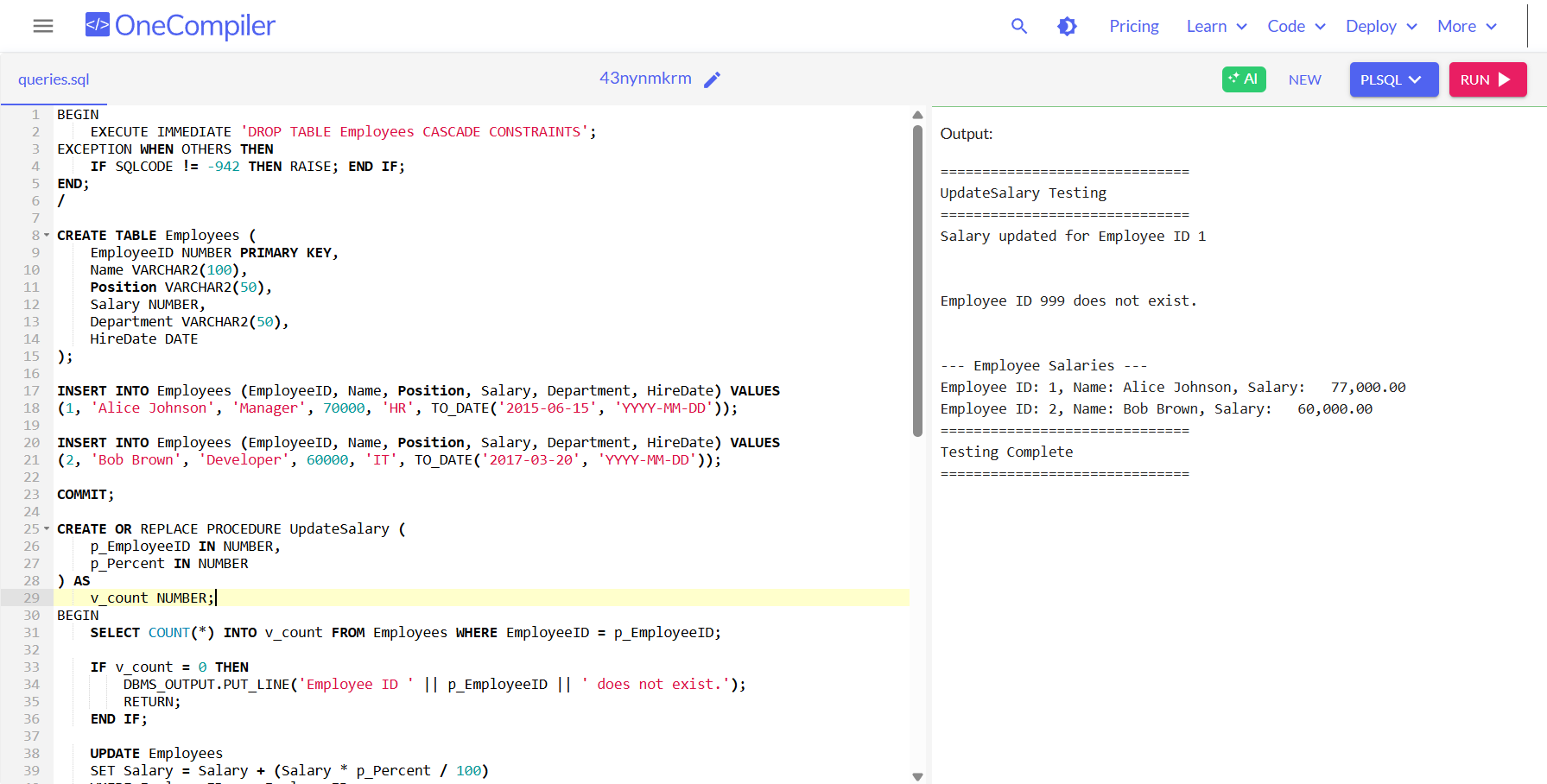
DBMS\_OUTPUT.PUT\_LINE(' Testing Complete ');

DBMS\_OUTPUT.PUT\_LINE('==============================');

END;

/

**Output:**

****

**Scenario 3:** Ensure data integrity when adding a new customer.

* + **Question:** Write a stored procedure **AddNewCustomer** that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

**Solution:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE ErrorLog CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE DEFAULT SYSDATE

);

CREATE TABLE ErrorLog (

LogID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

ErrorMessage VARCHAR2(400),

LoggedAt TIMESTAMP DEFAULT SYSTIMESTAMP

);

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_DOB IN DATE,

p\_Balance IN NUMBER

)

AS

v\_exists NUMBER;

v\_error VARCHAR2(400);

BEGIN

SELECT COUNT(\*) INTO v\_exists FROM Customers WHERE CustomerID = p\_CustomerID;

IF v\_exists > 0 THEN

-- Compose the error string separately

v\_error := 'Customer ID ' || TO\_CHAR(p\_CustomerID) || ' already exists. Insert skipped.';

INSERT INTO ErrorLog (ErrorMessage) VALUES (v\_error);

DBMS\_OUTPUT.PUT\_LINE(' ' || v\_error);

ELSE

INSERT INTO Customers (CustomerID, Name, DOB, Balance)

VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance);

DBMS\_OUTPUT.PUT\_LINE(' Customer "' || p\_Name || '" added successfully.');

END IF;

EXCEPTION

WHEN OTHERS THEN

v\_error := 'Error inserting customer ID ' || TO\_CHAR(p\_CustomerID) || ': ' || SQLERRM;

INSERT INTO ErrorLog (ErrorMessage) VALUES (v\_error);

DBMS\_OUTPUT.PUT\_LINE('' || v\_error);

END;

/

SET SERVEROUTPUT ON;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('==================== AddNewCustomer Test ====================');

-- Insert new customer

AddNewCustomer(101, 'Alice Brown', TO\_DATE('1992-03-05', 'YYYY-MM-DD'), 2000);

-- Attempt to insert same customer again

AddNewCustomer(101, 'Alice Brown', TO\_DATE('1992-03-05', 'YYYY-MM-DD'), 2000);

-- Insert another customer

AddNewCustomer(102, 'Bob Stone', TO\_DATE('1988-11-10', 'YYYY-MM-DD'), 2500);

-- Show Customers

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Customers Table ---');

FOR c IN (SELECT \* FROM Customers ORDER BY CustomerID) LOOP

DBMS\_OUTPUT.PUT\_LINE('ID: ' || c.CustomerID || ', Name: ' || c.Name || ', Balance: ' || c.Balance);

END LOOP;

-- Show Error Log

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Error Log ---');

FOR e IN (SELECT \* FROM ErrorLog ORDER BY LoggedAt DESC) LOOP

DBMS\_OUTPUT.PUT\_LINE(e.LoggedAt || ' | ' || e.ErrorMessage);

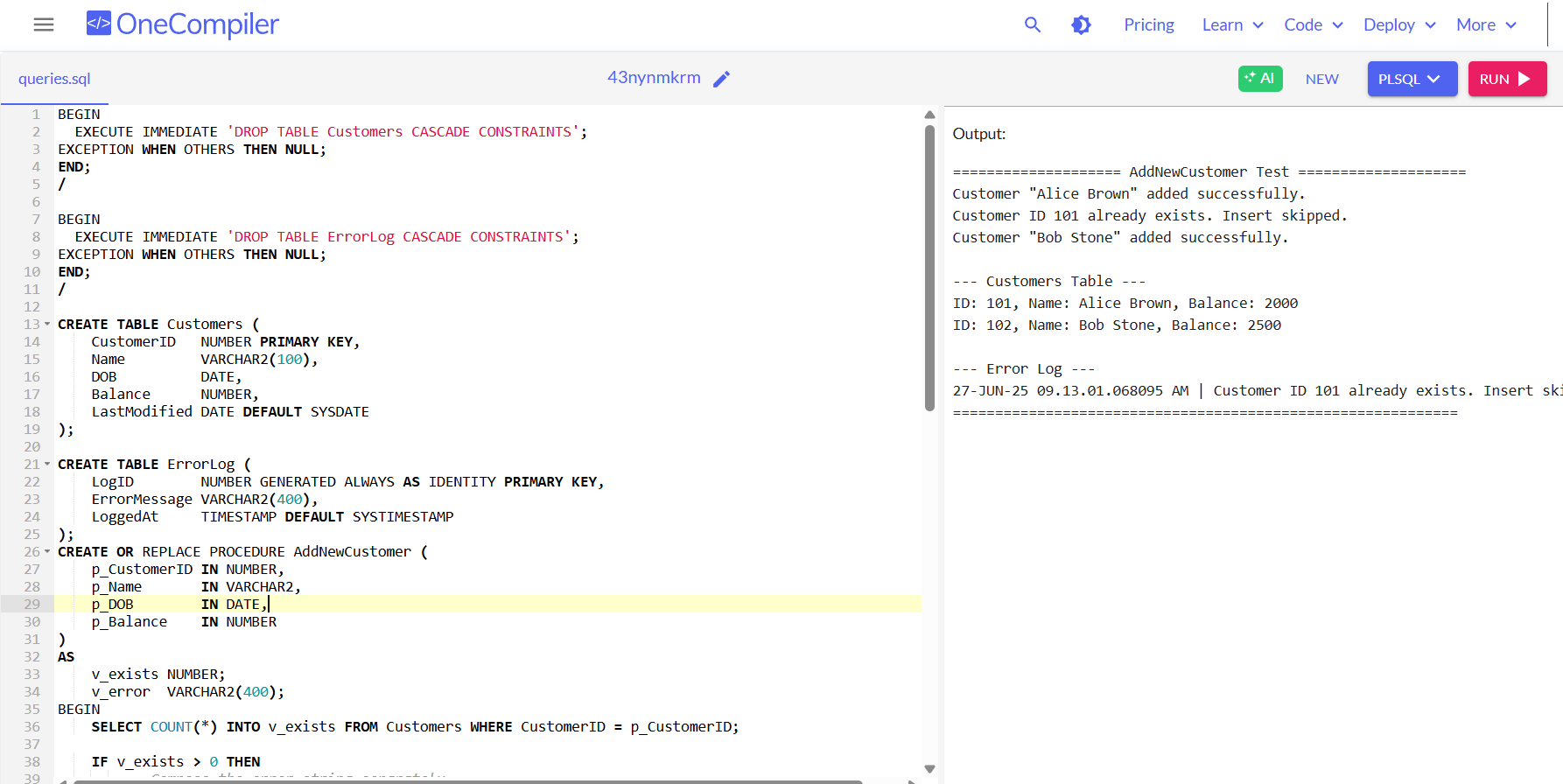
END LOOP;

DBMS\_OUTPUT.PUT\_LINE('============================================================');

END;

/

**Output:**

****

**Exercise 3: Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

* + **Question:** Write a stored procedure ProcessMonthlyInterest that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

**Solution:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

CONSTRAINT fk\_customer FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1985-05-15','YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20','YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

COMMIT;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Before applying interest:');

FOR rec IN (SELECT AccountID, CustomerID, AccountType, Balance FROM Accounts ORDER BY AccountID) LOOP

DBMS\_OUTPUT.PUT\_LINE('AccountID: ' || rec.AccountID ||

', CustomerID: ' || rec.CustomerID ||

', AccountType: ' || rec.AccountType ||

', Balance: ' || rec.Balance);

END LOOP;

END;

/

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01),

LastModified = SYSDATE

WHERE UPPER(TRIM(AccountType)) = 'SAVINGS';

COMMIT;

END;

/

BEGIN

ProcessMonthlyInterest;

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE('After applying interest:');

FOR rec IN (SELECT AccountID, CustomerID, AccountType, Balance FROM Accounts ORDER BY AccountID) LOOP

DBMS\_OUTPUT.PUT\_LINE('AccountID: ' || rec.AccountID ||

', CustomerID: ' || rec.CustomerID ||

', AccountType: ' || rec.AccountType ||

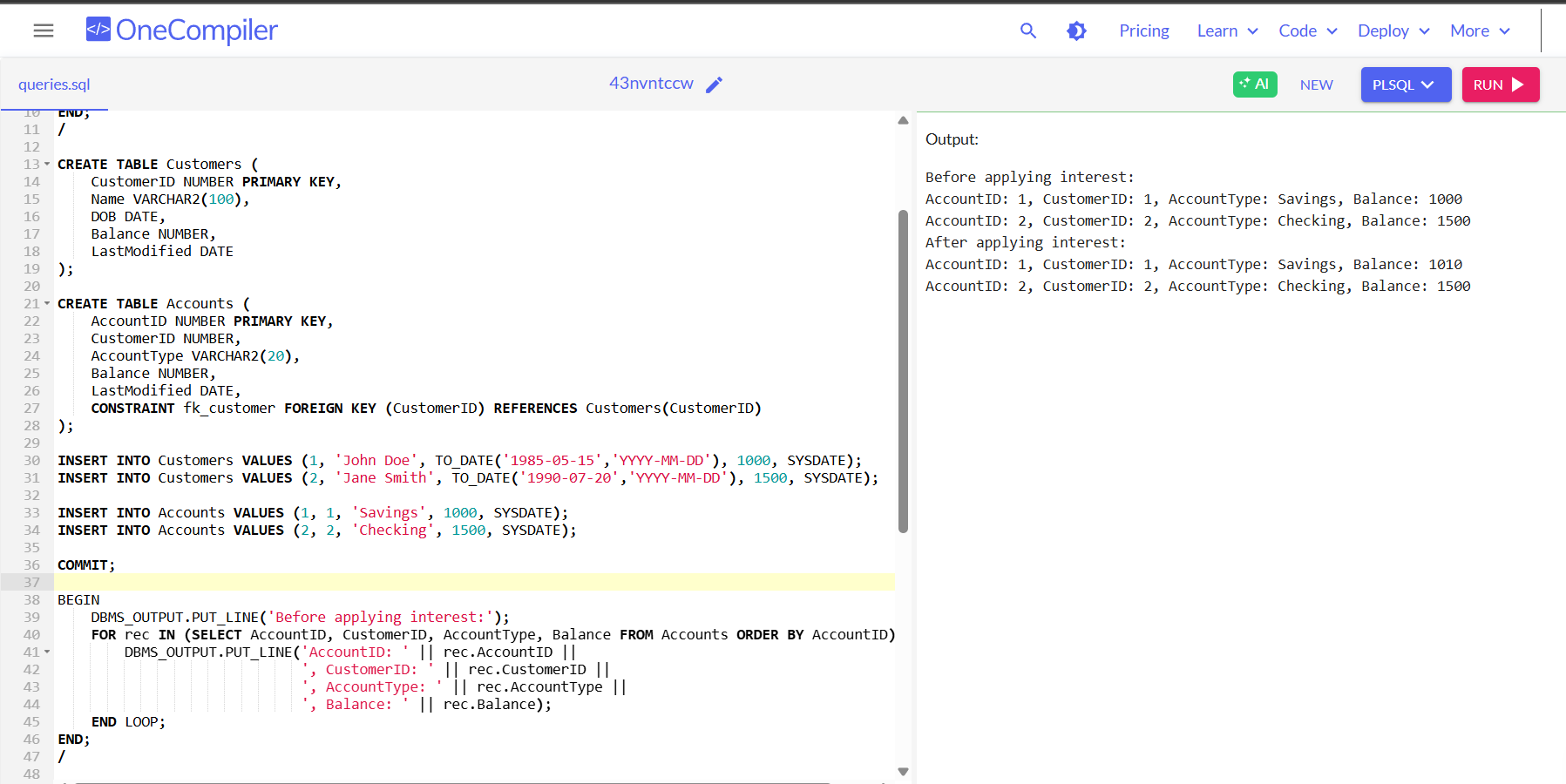
', Balance: ' || rec.Balance);

END LOOP;

END;

/

**Output:**

****

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

* + **Question:** Write a stored procedure UpdateEmployeeBonus that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

**Solution:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Employees CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DepartmentID NUMBER,

Salary NUMBER,

LastModified DATE

);

INSERT INTO Employees VALUES (1, 'Alice', 10, 50000, SYSDATE);

INSERT INTO Employees VALUES (2, 'Bob', 10, 60000, SYSDATE);

INSERT INTO Employees VALUES (3, 'Charlie', 20, 55000, SYSDATE);

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department\_id IN NUMBER,

p\_bonus\_pct IN NUMBER

) IS

BEGIN

-- Show BEFORE bonus

DBMS\_OUTPUT.PUT\_LINE('Before applying bonus:');

FOR rec IN (

SELECT EmployeeID, RPAD(Name, 10) AS Name,

TO\_CHAR(Salary, '999,999.00') AS Salary

FROM Employees

WHERE DepartmentID = p\_department\_id

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee: ' || rec.Name || ' | ID: ' || rec.EmployeeID || ' | Salary: ' || rec.Salary);

END LOOP;

-- Apply bonus

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_pct / 100),

LastModified = SYSDATE

WHERE DepartmentID = p\_department\_id;

COMMIT;

-- Show AFTER bonus

DBMS\_OUTPUT.PUT\_LINE('After applying bonus:');

FOR rec IN (

SELECT EmployeeID, RPAD(Name, 10) AS Name,

TO\_CHAR(Salary, '999,999.00') AS Salary

FROM Employees

WHERE DepartmentID = p\_department\_id

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee: ' || rec.Name || ' | ID: ' || rec.EmployeeID || ' | New Salary: ' || rec.Salary);

END LOOP;

END;

/

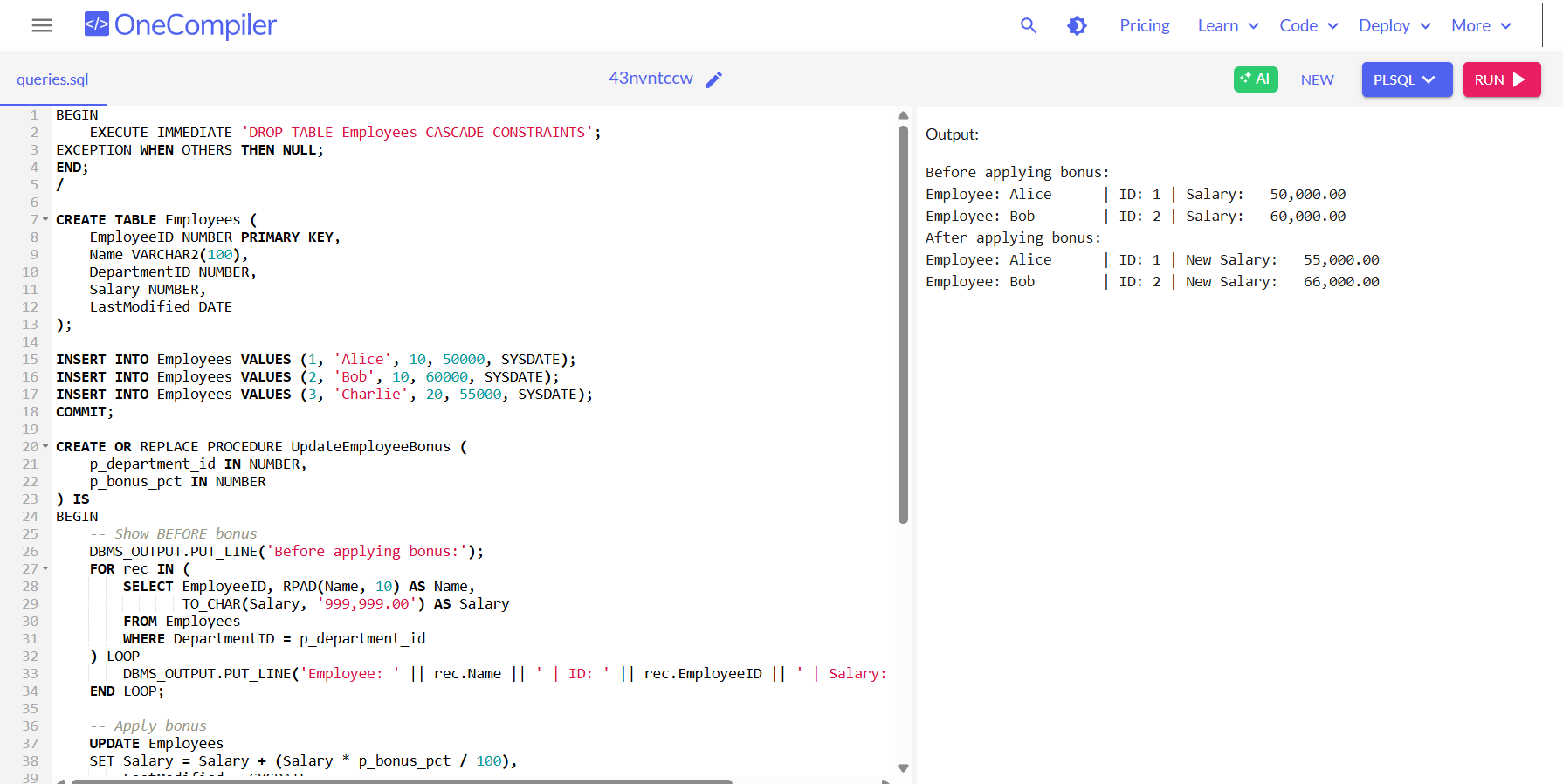
BEGIN

UpdateEmployeeBonus(10, 10);

END;

/

**Output:**

****

**Scenario 3:** Customers should be able to transfer funds between their accounts.

* + **Question:** Write a stored procedure TransferFunds that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

**Solution:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER,

LastModified DATE

);

INSERT INTO Accounts VALUES (1, 101, 5000, SYSDATE);

INSERT INTO Accounts VALUES (2, 102, 3000, SYSDATE);

COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_from\_balance NUMBER;

BEGIN

-- Show BEFORE transfer

DBMS\_OUTPUT.PUT\_LINE('Before transfer:');

FOR rec IN (

SELECT AccountID, TO\_CHAR(Balance, '999,999.00') AS Balance

FROM Accounts

WHERE AccountID IN (p\_from\_account, p\_to\_account)

ORDER BY AccountID

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Account ' || rec.AccountID || ' balance: ' || rec.Balance);

END LOOP;

-- Get source account balance

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account

FOR UPDATE;

-- Check funds

IF v\_from\_balance < p\_amount THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient funds in account ' || p\_from\_account);

RETURN;

END IF;

-- Perform transfer

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful: ' || p\_amount || ' transferred from account ' || p\_from\_account || ' to account ' || p\_to\_account);

-- Show AFTER transfer

DBMS\_OUTPUT.PUT\_LINE('After transfer:');

FOR rec IN (

SELECT AccountID, TO\_CHAR(Balance, '999,999.00') AS Balance

FROM Accounts

WHERE AccountID IN (p\_from\_account, p\_to\_account)

ORDER BY AccountID

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Account ' || rec.AccountID || ' balance: ' || rec.Balance);

END LOOP;

END;

/

BEGIN

TransferFunds(1, 2, 1000);

END;

/

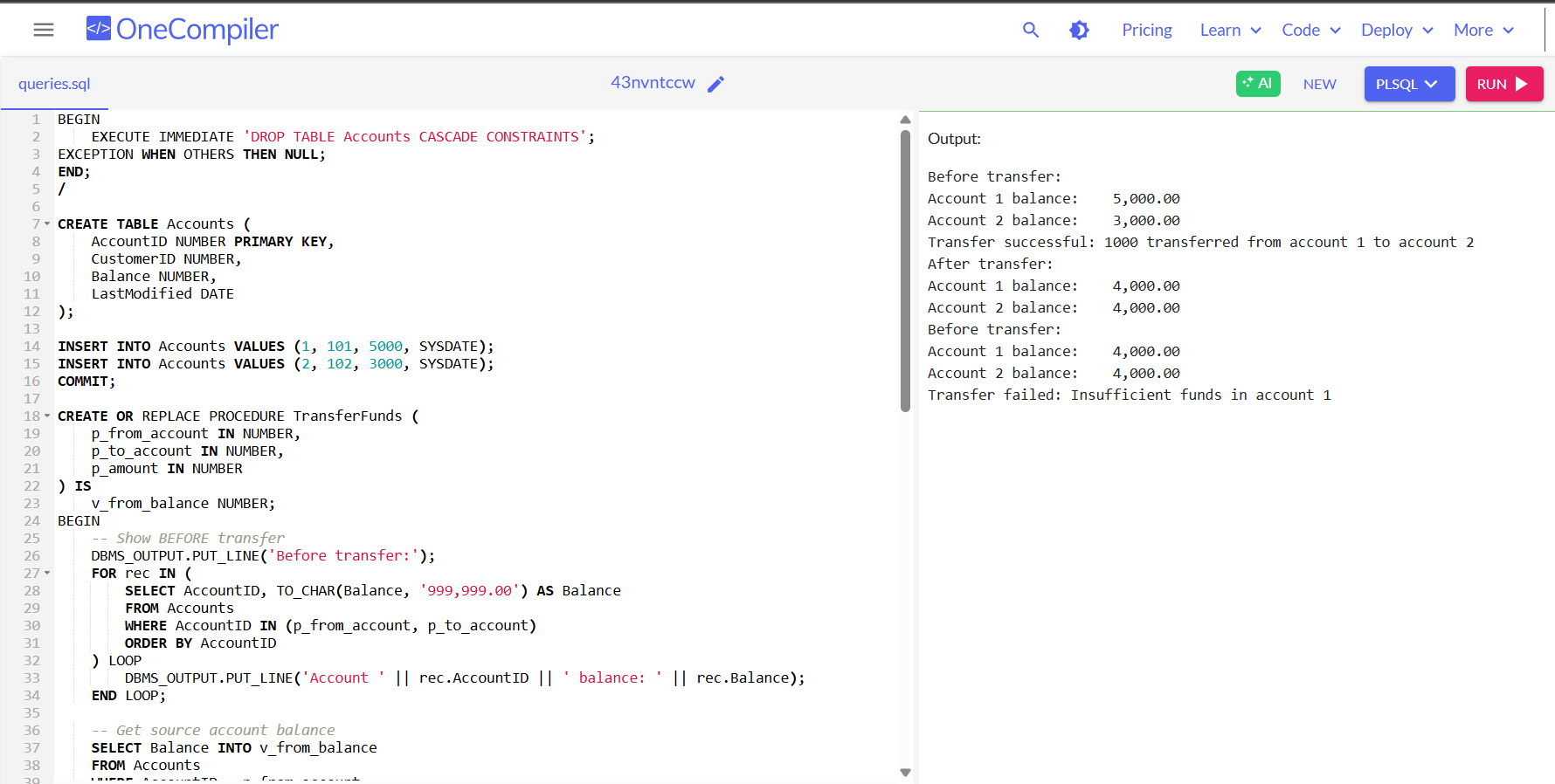
BEGIN

TransferFunds(1, 2, 10000);

END;

/

**Output:**

****

**Exercise 4: Functions**

**Scenario 1:** Calculate the age of customers for eligibility checks.

* + **Question:** Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years**.**

**Solution:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Loans CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Employees CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN

NULL; -- ignore errors if tables don't exist

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (3, 'Michael Johnson', TO\_DATE('1978-12-03', 'YYYY-MM-DD'), 2500, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (4, 'Emily Davis', TO\_DATE('2000-09-15', 'YYYY-MM-DD'), 1200, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (3, 3, 'Savings', 2500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (4, 4, 'Checking', 1200, SYSDATE);

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (3, 3, SYSDATE, 400, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (4, 4, SYSDATE, 150, 'Deposit');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 3, 10000, 6, SYSDATE, ADD\_MONTHS(SYSDATE, 120));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

COMMIT;

CREATE OR REPLACE FUNCTION CalculateAge (p\_dob DATE) RETURN NUMBER IS

v\_age NUMBER;

BEGIN

v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

RETURN v\_age;

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE('====== Customer DOB and Age Details ======');

DBMS\_OUTPUT.PUT\_LINE('Customer Name | Customer ID | DOB (Before) | Age (After Calculation)');

DBMS\_OUTPUT.PUT\_LINE('---------------------------------------------------------------');

FOR rec IN (SELECT CustomerID, Name, DOB FROM Customers ORDER BY CustomerID) LOOP

DBMS\_OUTPUT.PUT\_LINE(

rec.Name || ' | ' ||

rec.CustomerID || ' | ' ||

TO\_CHAR(rec.DOB, 'DD-MON-YYYY') || ' | ' ||

CalculateAge(rec.DOB) || ' years'

);

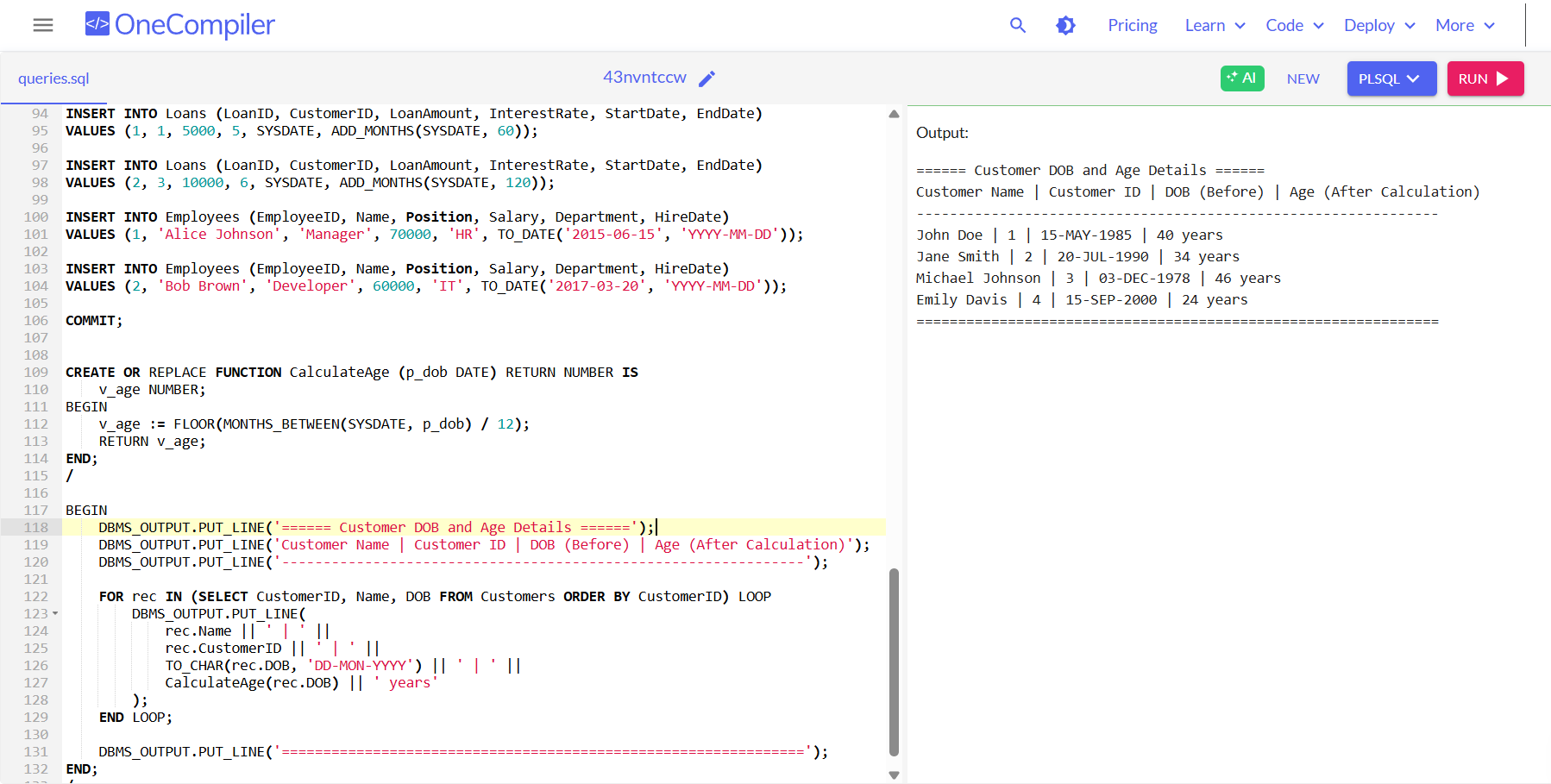
END LOOP;

DBMS\_OUTPUT.PUT\_LINE('===============================================================');

END;

/

**Output:**

****

**Scenario 2**: The bank needs to compute the monthly installment for a loan.

* + **Question:** Write a function CalculateMonthlyInstallment that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

**Solution:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER, -- Annual interest rate in percent, e.g. 5 for 5%

p\_loan\_years NUMBER

) RETURN NUMBER IS

v\_monthly\_rate NUMBER;

v\_total\_payments NUMBER;

v\_installment NUMBER;

BEGIN

-- Convert annual interest rate (%) to monthly decimal rate

v\_monthly\_rate := (p\_interest\_rate / 100) / 12;

-- Total number of monthly payments

v\_total\_payments := p\_loan\_years \* 12;

IF v\_monthly\_rate = 0 THEN

-- If interest rate is zero, installment is principal divided equally

v\_installment := p\_loan\_amount / v\_total\_payments;

ELSE

-- Use amortization formula to calculate installment

v\_installment := p\_loan\_amount \* (v\_monthly\_rate \* POWER(1 + v\_monthly\_rate, v\_total\_payments)) /

(POWER(1 + v\_monthly\_rate, v\_total\_payments) - 1);

END IF;

-- Round installment to 2 decimal places and return

RETURN ROUND(v\_installment, 2);

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE('================ Loan Monthly Installment Calculation ================');

DBMS\_OUTPUT.PUT\_LINE('Loan Amount | Interest Rate (Annual %) | Duration (Years) | Monthly Installment');

DBMS\_OUTPUT.PUT\_LINE('----------------------------------------------------------------------');

-- Sample 1

DBMS\_OUTPUT.PUT\_LINE(

TO\_CHAR(5000, '999,999.00') || ' | ' ||

TO\_CHAR(5, '990.00') || ' | ' ||

TO\_CHAR(5, '990') || ' | ' ||

TO\_CHAR(CalculateMonthlyInstallment(5000, 5, 5), '9999.99')

);

-- Sample 2

DBMS\_OUTPUT.PUT\_LINE(

TO\_CHAR(10000, '999,999.00') || ' | ' ||

TO\_CHAR(6, '990.00') || ' | ' ||

TO\_CHAR(10, '990') || ' | ' ||

TO\_CHAR(CalculateMonthlyInstallment(10000, 6, 10), '9999.99')

);

-- Sample 3 (Zero interest)

DBMS\_OUTPUT.PUT\_LINE(

TO\_CHAR(20000, '999,999.00') || ' | ' ||

TO\_CHAR(0, '990.00') || ' | ' ||

TO\_CHAR(4, '990') || ' | ' ||

TO\_CHAR(CalculateMonthlyInstallment(20000, 0, 4), '9999.99')

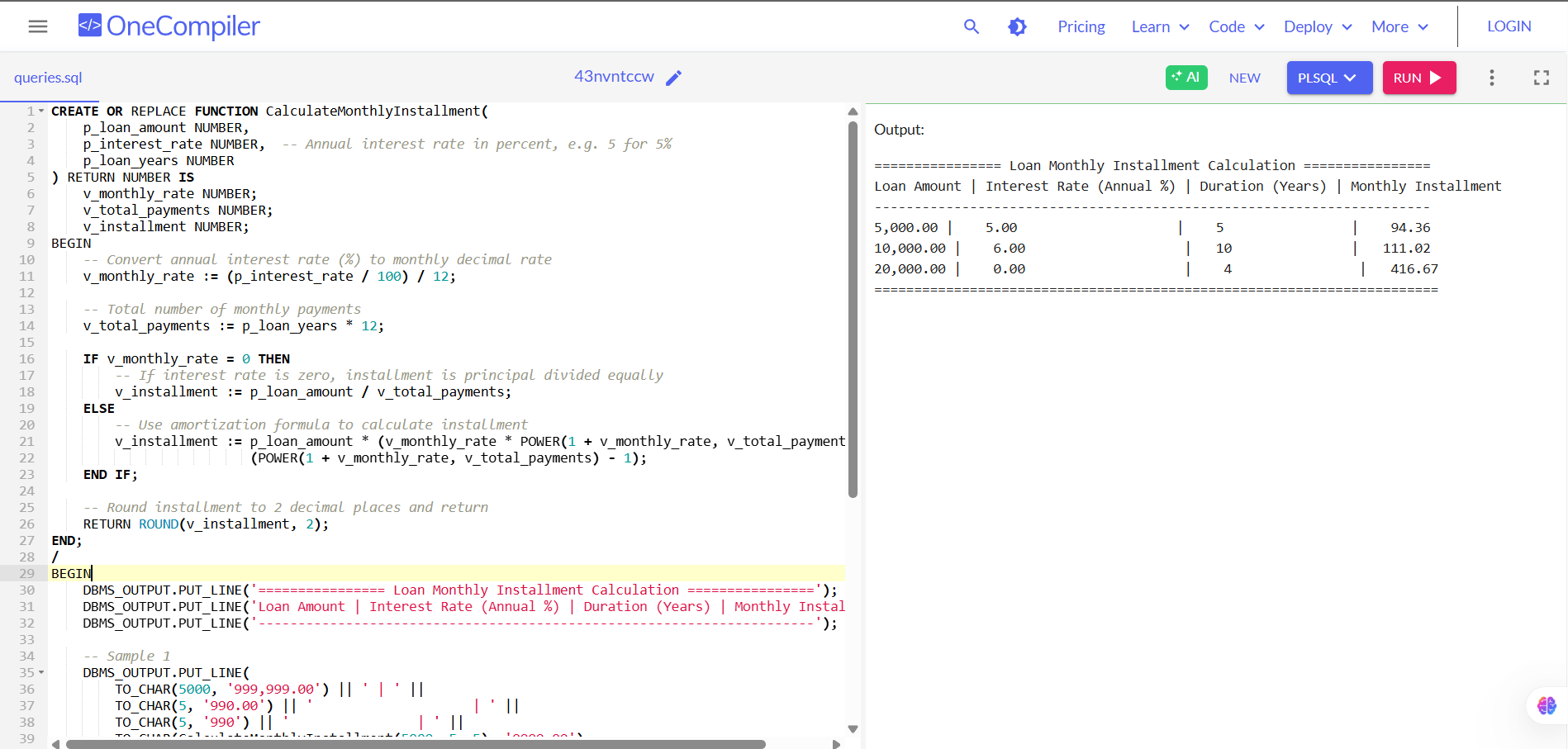
);

DBMS\_OUTPUT.PUT\_LINE('=======================================================================');

END;

/

**Output:**



**Scenario 3:** Check if a customer has sufficient balance before making a transaction.

* + **Question:** Write a function HasSufficientBalance that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

**Solution:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSDATE);

COMMIT;

CREATE OR REPLACE FUNCTION HasSufficientBalance(

p\_account\_id NUMBER,

p\_amount NUMBER

) RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_account\_id;

RETURN (v\_balance >= p\_amount);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN RETURN FALSE;

WHEN OTHERS THEN RETURN FALSE;

END;

/

CREATE OR REPLACE PROCEDURE WithdrawAmount(

p\_account\_id NUMBER,

p\_amount NUMBER

) IS

v\_balance\_before NUMBER;

v\_balance\_after NUMBER;

v\_customer\_name VARCHAR2(100);

BEGIN

-- Get customer name for nicer output

SELECT c.Name INTO v\_customer\_name

FROM Customers c

JOIN Accounts a ON c.CustomerID = a.CustomerID

WHERE a.AccountID = p\_account\_id;

-- Get balance before withdrawal

SELECT Balance INTO v\_balance\_before FROM Accounts WHERE AccountID = p\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('-----------------------------------------------------------');

DBMS\_OUTPUT.PUT\_LINE('Attempting Withdrawal for Account ID: ' || p\_account\_id);

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || v\_customer\_name);

DBMS\_OUTPUT.PUT\_LINE('Current Balance BEFORE withdrawal: ' || TO\_CHAR(v\_balance\_before, '999,999.99'));

DBMS\_OUTPUT.PUT\_LINE('Requested Withdrawal Amount: ' || TO\_CHAR(p\_amount, '999,999.99'));

IF HasSufficientBalance(p\_account\_id, p\_amount) THEN

-- Perform withdrawal

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_account\_id;

-- Get balance after withdrawal

SELECT Balance INTO v\_balance\_after FROM Accounts WHERE AccountID = p\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('Withdrawal Status: SUCCESS');

DBMS\_OUTPUT.PUT\_LINE('Balance AFTER withdrawal: ' || TO\_CHAR(v\_balance\_after, '999,999.99'));

ELSE

DBMS\_OUTPUT.PUT\_LINE('Withdrawal Status: FAILED - Insufficient balance');

END IF;

DBMS\_OUTPUT.PUT\_LINE('-----------------------------------------------------------');

END;

/

BEGIN

-- Test 1: Sufficient balance withdrawal

WithdrawAmount(1, 500);

-- Test 2: Insufficient balance withdrawal

WithdrawAmount(2, 2000);

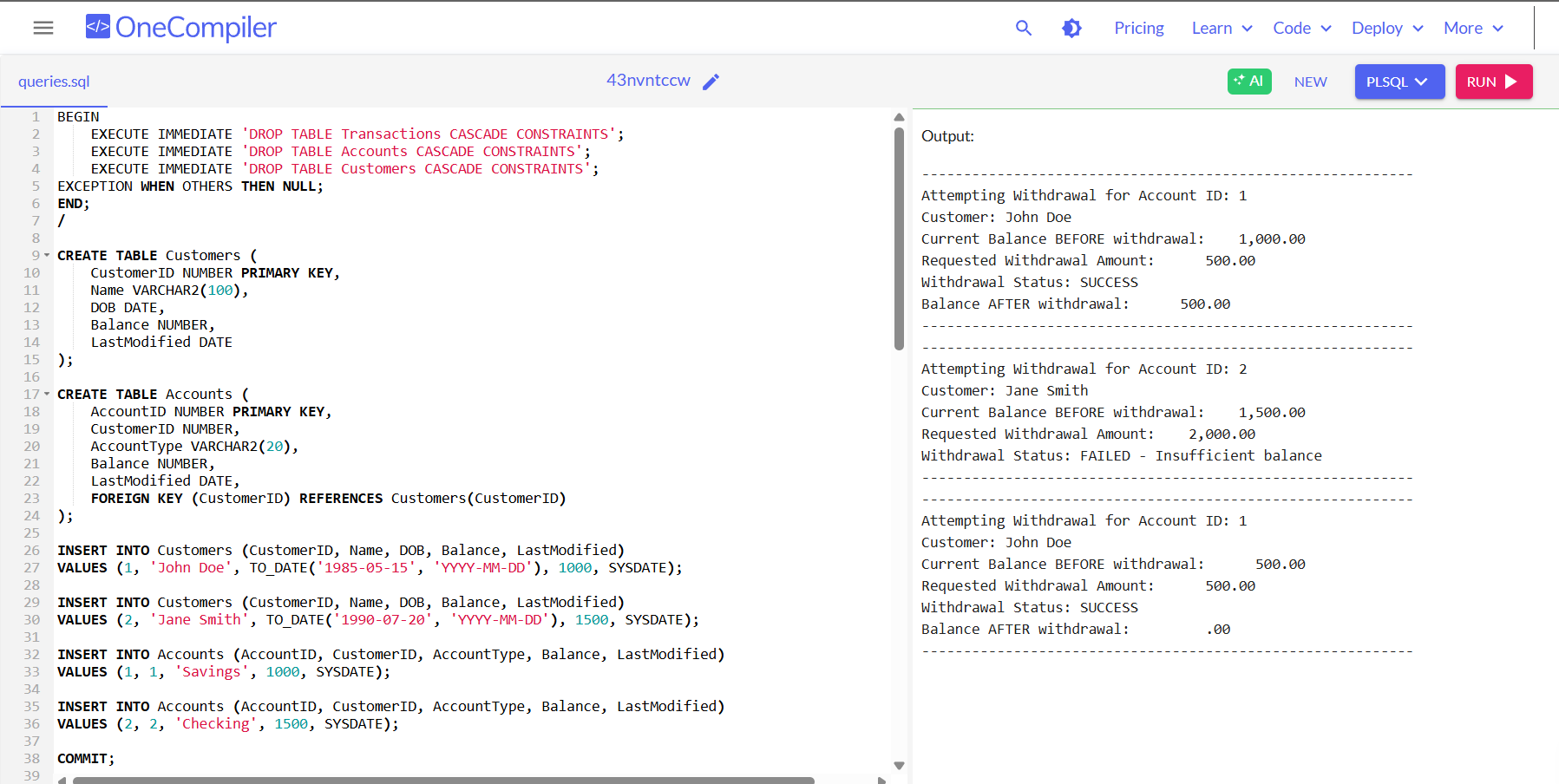
-- Test 3: Exact remaining balance withdrawal

WithdrawAmount(1, 500);

END;

/

**Output:**

****

**Exercise 5: Triggers**

**Scenario 1:** Automatically update the last modified date when a customer's record is updated.

* + **Question:** Write a trigger UpdateCustomerLastModified that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.

**Solution:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified TIMESTAMP

);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSTIMESTAMP);

COMMIT;

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSTIMESTAMP;

END;

/

DECLARE

v\_before TIMESTAMP;

v\_after TIMESTAMP;

v\_name VARCHAR2(100);

BEGIN

SELECT Name, LastModified INTO v\_name, v\_before

FROM Customers

WHERE CustomerID = 1;

DBMS\_OUTPUT.PUT\_LINE('===================================================');

DBMS\_OUTPUT.PUT\_LINE('Before Update');

DBMS\_OUTPUT.PUT\_LINE('Customer ID: 1');

DBMS\_OUTPUT.PUT\_LINE('Customer Name: ' || v\_name);

DBMS\_OUTPUT.PUT\_LINE('Last Modified: ' || TO\_CHAR(v\_before, 'DD-MON-YYYY HH24:MI:SS.FF3'));

DBMS\_OUTPUT.PUT\_LINE('===================================================');

-- Update

UPDATE Customers

SET Balance = Balance + 250

WHERE CustomerID = 1;

SELECT LastModified INTO v\_after

FROM Customers

WHERE CustomerID = 1;

DBMS\_OUTPUT.PUT\_LINE('===================================================');

DBMS\_OUTPUT.PUT\_LINE('After Update');

DBMS\_OUTPUT.PUT\_LINE('Customer ID: 1');

DBMS\_OUTPUT.PUT\_LINE('Customer Name: ' || v\_name);

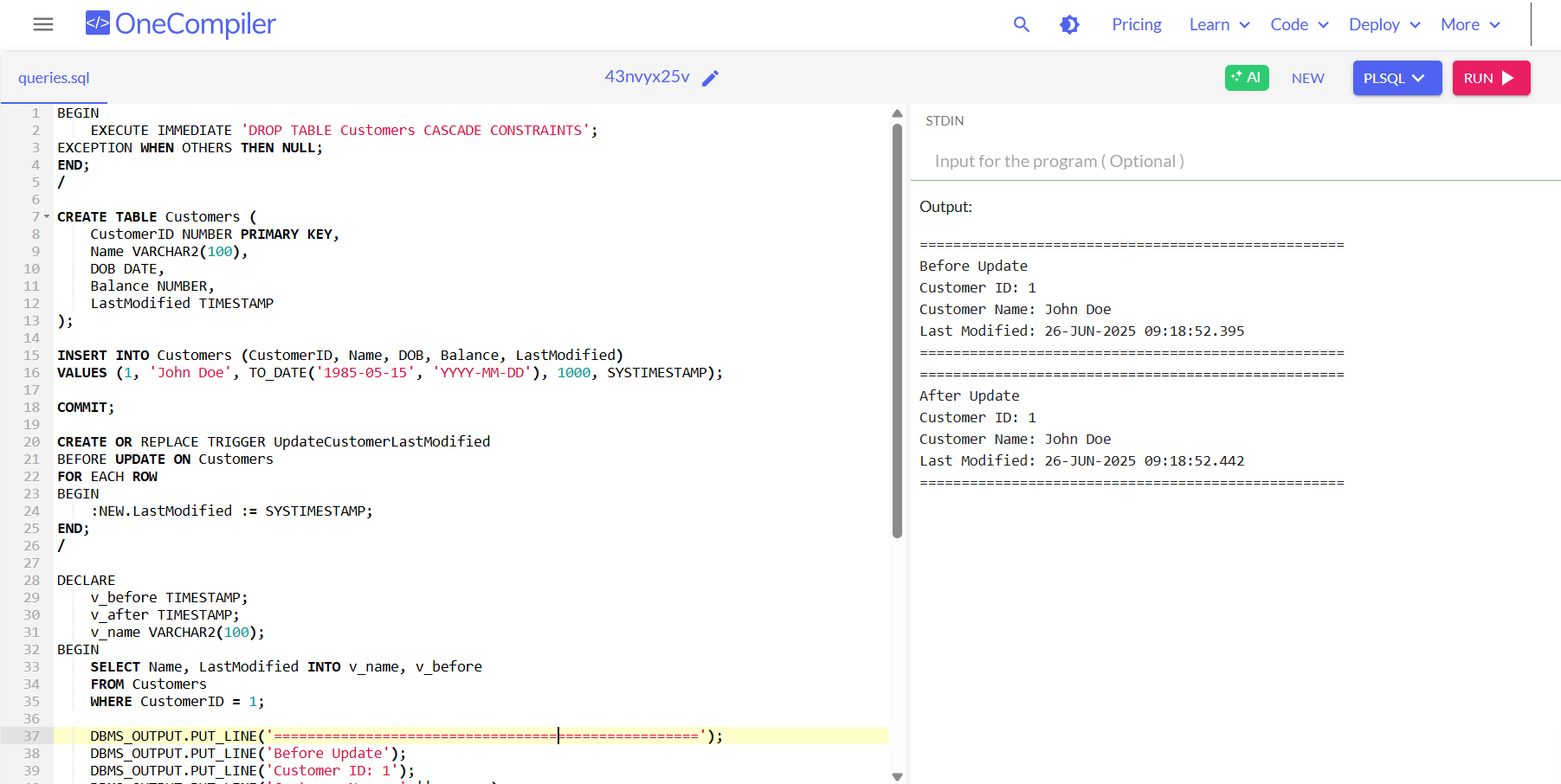
DBMS\_OUTPUT.PUT\_LINE('Last Modified: ' || TO\_CHAR(v\_after, 'DD-MON-YYYY HH24:MI:SS.FF3'));

DBMS\_OUTPUT.PUT\_LINE('===================================================');

END;

/

**Output:**

****

**Scenario 2:** Maintain an audit log for all transactions.

* + **Question:** Write a trigger LogTransaction that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

**Solution:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE AuditLog CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Transactions CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified TIMESTAMP

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified TIMESTAMP,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate TIMESTAMP,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE AuditLog (

LogID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

Action VARCHAR2(20),

ActionDate TIMESTAMP

);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSTIMESTAMP);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSTIMESTAMP);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSTIMESTAMP);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSTIMESTAMP);

COMMIT;

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, AccountID, Action, ActionDate)

VALUES (:NEW.TransactionID, :NEW.AccountID, 'INSERT', SYSTIMESTAMP);

END;

/

DECLARE

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=============== BEFORE INSERT ===============');

-- Show Transactions table

FOR rec IN (

SELECT TransactionID, AccountID, Amount, TransactionType,

TO\_CHAR(TransactionDate, 'DD-MON-YYYY HH24:MI:SS.FF3') AS TransactionDate

FROM Transactions

ORDER BY TransactionID

) LOOP

DBMS\_OUTPUT.PUT\_LINE('TransactionID: ' || rec.TransactionID ||

' | AccountID: ' || rec.AccountID ||

' | Amount: ' || rec.Amount ||

' | Type: ' || rec.TransactionType ||

' | Date: ' || rec.TransactionDate);

END LOOP;

-- Show AuditLog table

DBMS\_OUTPUT.PUT\_LINE('--------------- AUDIT LOG ------------------');

FOR rec IN (

SELECT LogID, TransactionID, AccountID, Action,

TO\_CHAR(ActionDate, 'DD-MON-YYYY HH24:MI:SS.FF3') AS ActionDate

FROM AuditLog

ORDER BY LogID

) LOOP

DBMS\_OUTPUT.PUT\_LINE('LogID: ' || rec.LogID ||

' | TransactionID: ' || rec.TransactionID ||

' | AccountID: ' || rec.AccountID ||

' | Action: ' || rec.Action ||

' | Date: ' || rec.ActionDate);

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('=============================================');

END;

/

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (1, 1, SYSTIMESTAMP, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (2, 2, SYSTIMESTAMP, 300, 'Withdrawal');

COMMIT;

-- Show AFTER insert state

DECLARE

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=============== AFTER INSERT ================');

-- Show Transactions table

FOR rec IN (

SELECT TransactionID, AccountID, Amount, TransactionType,

TO\_CHAR(TransactionDate, 'DD-MON-YYYY HH24:MI:SS.FF3') AS TransactionDate

FROM Transactions

ORDER BY TransactionID

) LOOP

DBMS\_OUTPUT.PUT\_LINE('TransactionID: ' || rec.TransactionID ||

' | AccountID: ' || rec.AccountID ||

' | Amount: ' || rec.Amount ||

' | Type: ' || rec.TransactionType ||

' | Date: ' || rec.TransactionDate);

END LOOP;

-- Show AuditLog table

DBMS\_OUTPUT.PUT\_LINE('--------------- AUDIT LOG ------------------');

FOR rec IN (

SELECT LogID, TransactionID, AccountID, Action,

TO\_CHAR(ActionDate, 'DD-MON-YYYY HH24:MI:SS.FF3') AS ActionDate

FROM AuditLog

ORDER BY LogID

) LOOP

DBMS\_OUTPUT.PUT\_LINE('LogID: ' || rec.LogID ||

' | TransactionID: ' || rec.TransactionID ||

' | AccountID: ' || rec.AccountID ||

' | Action: ' || rec.Action ||

' | Date: ' || rec.ActionDate);

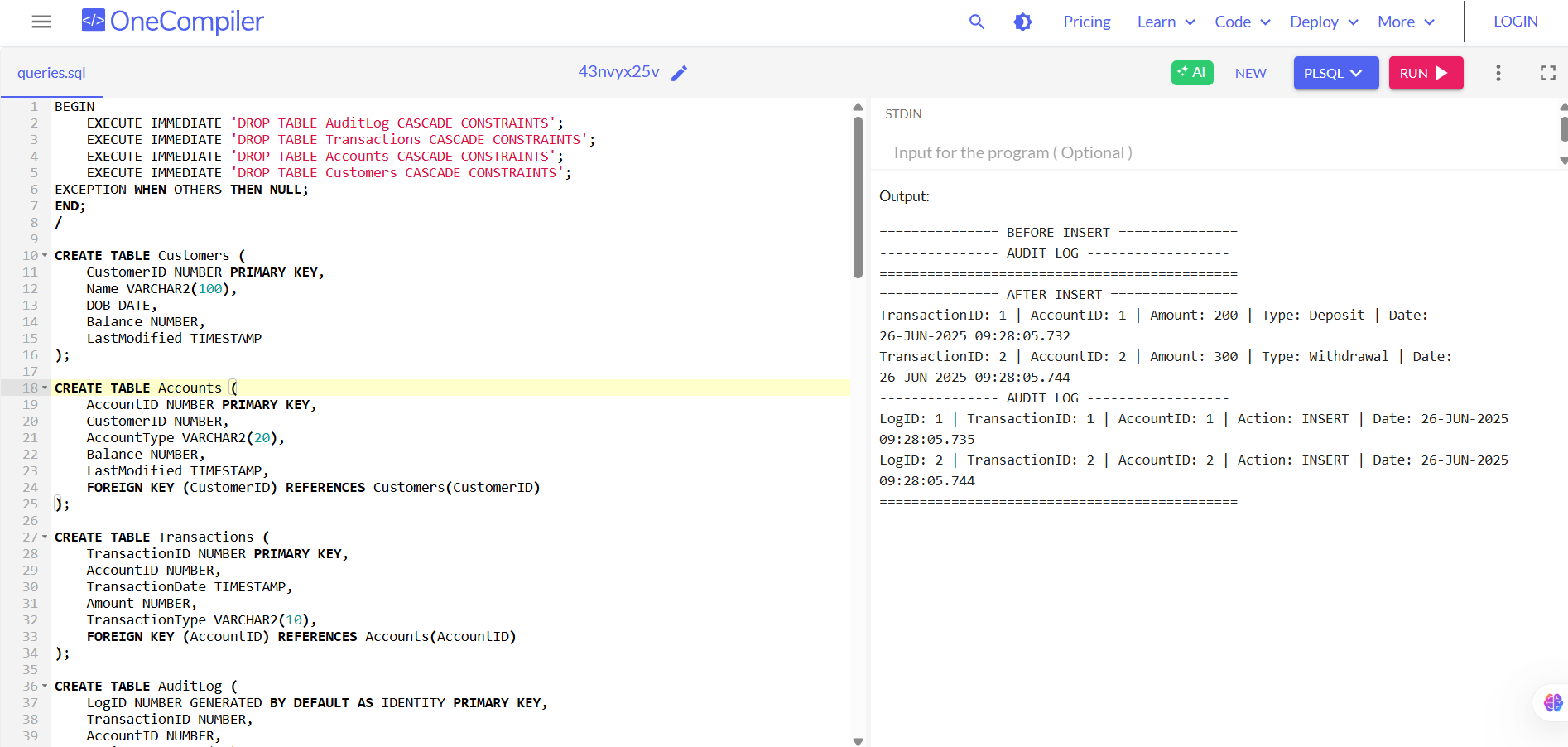
END LOOP;

DBMS\_OUTPUT.PUT\_LINE('=============================================');

END;

/

**Output:**

****

**Scenario 3:** Enforce business rules on deposits and withdrawals.

* + **Question:** Write a trigger CheckTransactionRules that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

**Solution:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified TIMESTAMP

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified TIMESTAMP,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate TIMESTAMP,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSTIMESTAMP);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSTIMESTAMP);

COMMIT;

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = :NEW.AccountID;

IF :NEW.TransactionType = 'Withdrawal' THEN

IF :NEW.Amount > v\_balance THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Withdrawal exceeds account balance.');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Deposit amount must be positive.');

END IF;

END IF;

END;

/

DECLARE

PROCEDURE try\_insert\_transaction(p\_trans\_id NUMBER, p\_acc\_id NUMBER, p\_amt NUMBER, p\_type VARCHAR2) IS

v\_full\_err\_msg VARCHAR2(4000);

v\_clean\_msg VARCHAR2(4000);

v\_pos NUMBER;

BEGIN

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (p\_trans\_id, p\_acc\_id, SYSTIMESTAMP, p\_amt, p\_type);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('SUCCESS: ' || p\_type || ' of ' || p\_amt || ' inserted for Account ' || p\_acc\_id);

EXCEPTION

WHEN OTHERS THEN

v\_full\_err\_msg := SQLERRM;

v\_pos := INSTR(v\_full\_err\_msg, CHR(10));

IF v\_pos > 0 THEN

v\_clean\_msg := SUBSTR(v\_full\_err\_msg, 1, v\_pos - 1);

ELSE

v\_clean\_msg := v\_full\_err\_msg;

END IF;

IF SQLCODE BETWEEN -20999 AND -20000 THEN

DBMS\_OUTPUT.PUT\_LINE('FAILURE: ' || p\_type || ' of ' || p\_amt || ' for Account ' || p\_acc\_id || ' -> ' || v\_clean\_msg);

ELSE

DBMS\_OUTPUT.PUT\_LINE('FAILURE: ' || p\_type || ' of ' || p\_amt || ' for Account ' || p\_acc\_id || ' -> Unexpected error: ' || v\_clean\_msg);

END IF;

END;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('================ Transaction Test ================');

try\_insert\_transaction(1, 1, 500, 'Deposit');

try\_insert\_transaction(2, 1, 0, 'Deposit');

try\_insert\_transaction(3, 1, 300, 'Withdrawal');

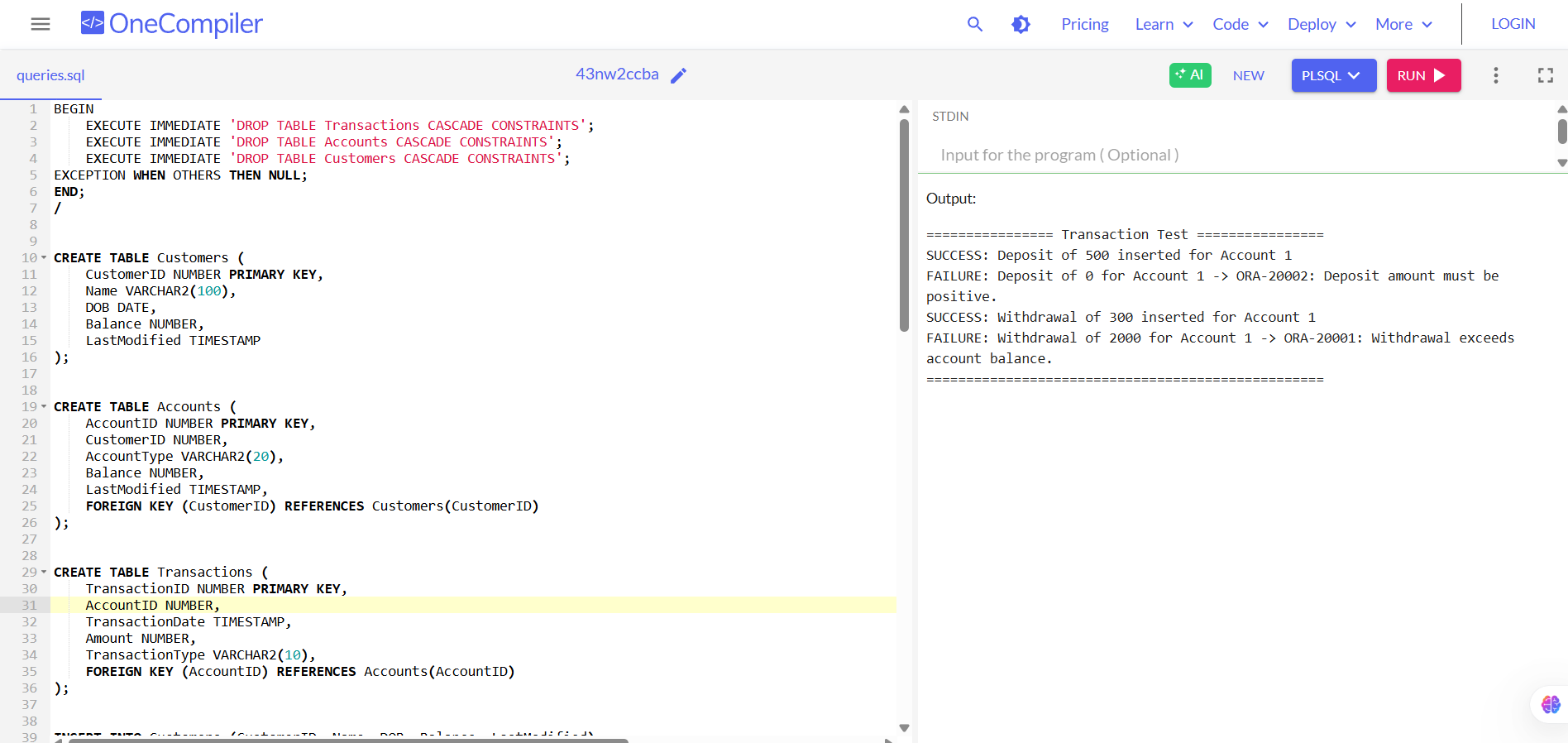
try\_insert\_transaction(4, 1, 2000, 'Withdrawal');

DBMS\_OUTPUT.PUT\_LINE('==================================================');

END;

/

**Output:**

****

**Exercise 6: Cursors**

**Scenario 1:** Generate monthly statements for all customers.

* + **Question:** Write a PL/SQL block using an explicit cursor GenerateMonthlyStatements that retrieves all transactions for the current month and prints a statement for each customer.

**Solution:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1985-05-15','YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20','YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Transactions VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions VALUES (2, 1, SYSDATE, 50, 'Withdrawal');

INSERT INTO Transactions VALUES (3, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Transactions VALUES (4, 2, ADD\_MONTHS(SYSDATE, -1), 150, 'Deposit'); -- last month

COMMIT;

SET SERVEROUTPUT ON;

DECLARE

CURSOR cur\_statements IS

SELECT

c.CustomerID,

c.Name AS CustomerName,

a.AccountID,

t.TransactionDate,

t.TransactionType,

t.Amount

FROM Customers c

JOIN Accounts a ON c.CustomerID = a.CustomerID

JOIN Transactions t ON a.AccountID = t.AccountID

WHERE TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM')

ORDER BY c.CustomerID, t.TransactionDate;

rec cur\_statements%ROWTYPE;

last\_cust\_id NUMBER := NULL;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=========== Monthly Statement ===========');

OPEN cur\_statements;

LOOP

FETCH cur\_statements INTO rec;

EXIT WHEN cur\_statements%NOTFOUND;

IF last\_cust\_id IS NULL OR last\_cust\_id != rec.CustomerID THEN

DBMS\_OUTPUT.PUT\_LINE('------------------------------------------');

DBMS\_OUTPUT.PUT\_LINE('Customer ID : ' || rec.CustomerID);

DBMS\_OUTPUT.PUT\_LINE('Customer Name : ' || rec.CustomerName);

DBMS\_OUTPUT.PUT\_LINE('Account ID : ' || rec.AccountID);

DBMS\_OUTPUT.PUT\_LINE('Date | Type | Amount');

DBMS\_OUTPUT.PUT\_LINE('------------|------------|--------');

last\_cust\_id := rec.CustomerID;

END IF;

DBMS\_OUTPUT.PUT\_LINE(

TO\_CHAR(rec.TransactionDate, 'DD-MON-YYYY') || ' | ' ||

RPAD(rec.TransactionType, 10) || ' | ' ||

TO\_CHAR(rec.Amount, '9990.00')

);

END LOOP;

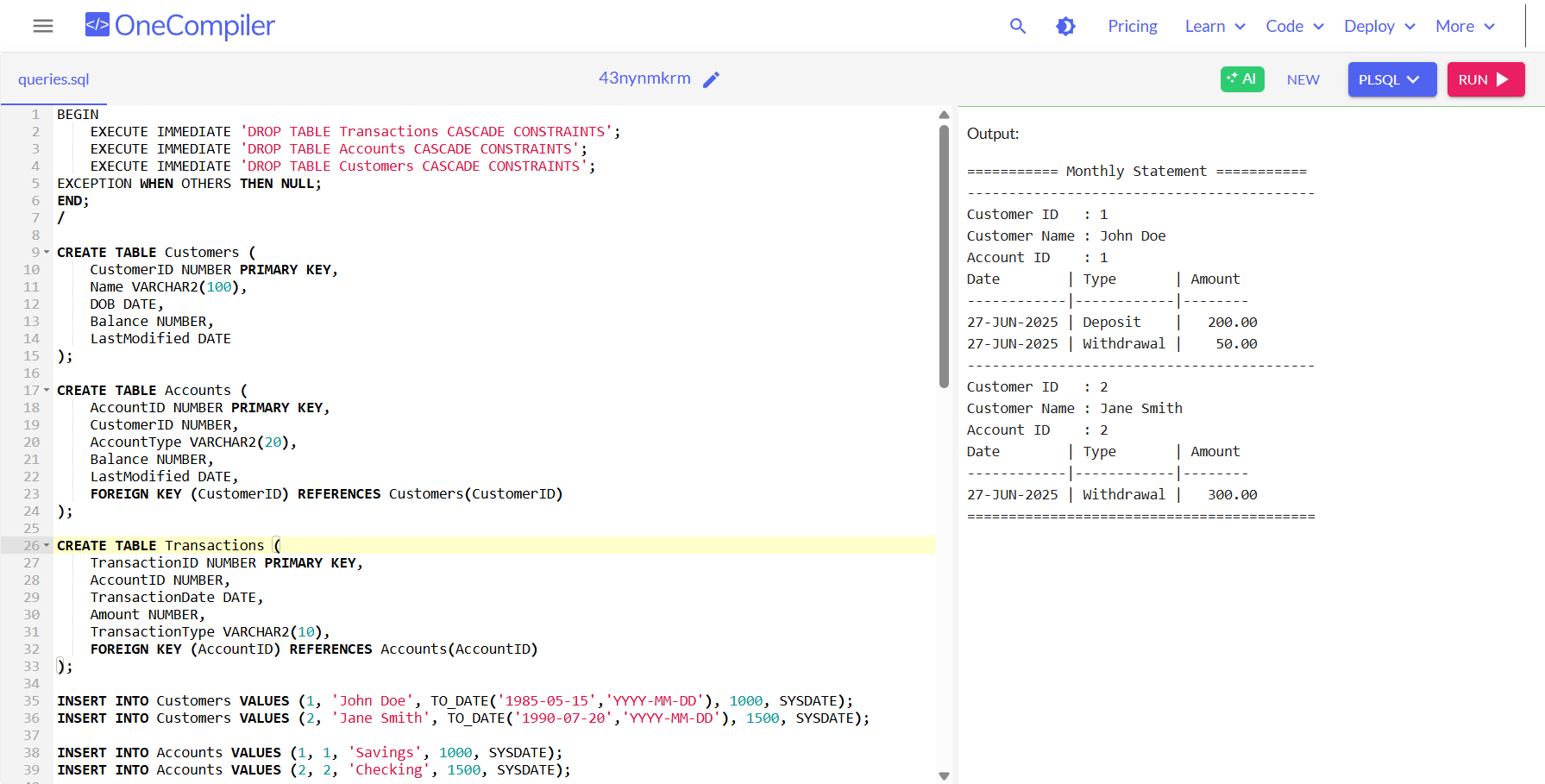
CLOSE cur\_statements;

DBMS\_OUTPUT.PUT\_LINE('==========================================');

END;

/

**Output:**

****

**Scenario 2:** Apply annual fee to all accounts.

* + **Question:** Write a PL/SQL block using an explicit cursor ApplyAnnualFee that deducts an annual maintenance fee from the balance of all accounts.

**Solution:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

COMMIT;

SET SERVEROUTPUT ON;

DECLARE

CURSOR cur\_accounts IS

SELECT AccountID, Balance

FROM Accounts;

rec\_account cur\_accounts%ROWTYPE;

v\_annual\_fee CONSTANT NUMBER := 100;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('========= Applying Annual Maintenance Fee =========');

OPEN cur\_accounts;

LOOP

FETCH cur\_accounts INTO rec\_account;

EXIT WHEN cur\_accounts%NOTFOUND;

-- Show balance before fee

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || rec\_account.AccountID ||

' | Balance Before: ' || rec\_account.Balance);

-- Deduct fee

UPDATE Accounts

SET Balance = Balance - v\_annual\_fee,

LastModified = SYSDATE

WHERE AccountID = rec\_account.AccountID;

-- Show updated balance

DECLARE

v\_new\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_new\_balance

FROM Accounts

WHERE AccountID = rec\_account.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Annual Fee of ' || v\_annual\_fee || ' deducted.');

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || rec\_account.AccountID ||

' | Balance After: ' || v\_new\_balance);

DBMS\_OUTPUT.PUT\_LINE('------------------------------------------------');

END;

END LOOP;

CLOSE cur\_accounts;

COMMIT;

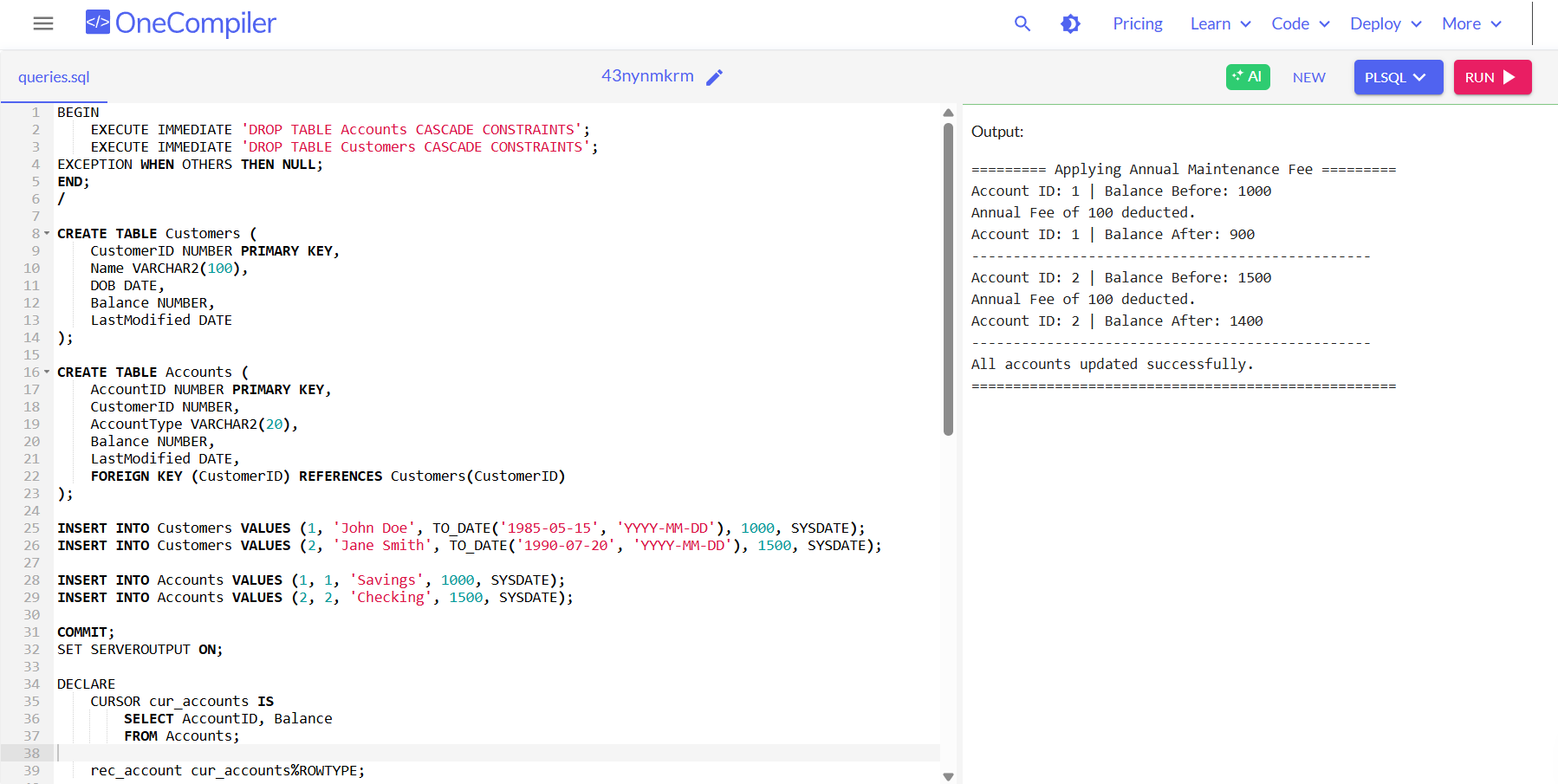
DBMS\_OUTPUT.PUT\_LINE(' All accounts updated successfully.');

DBMS\_OUTPUT.PUT\_LINE('===================================================');

END;

/

**Output:**

****

**Scenario 3:** Update the interest rate for all loans based on a new policy.

* + **Question:** Write a PL/SQL block using an explicit cursor UpdateLoanInterestRates that fetches all loans and updates their interest rates based on the new policy.

**Solution:**

SET SERVEROUTPUT ON;

DECLARE

-- Cursor to fetch all loans

CURSOR cur\_loans IS

SELECT LoanID, InterestRate

FROM Loans;

rec\_loan cur\_loans%ROWTYPE;

v\_new\_rate NUMBER;

BEGIN

-- 📝 Print policy details

DBMS\_OUTPUT.PUT\_LINE('========== Loan Interest Rate Update ==========');

DBMS\_OUTPUT.PUT\_LINE(' Policy Applied:');

DBMS\_OUTPUT.PUT\_LINE('- If InterestRate < 5%, increase by 1%');

DBMS\_OUTPUT.PUT\_LINE('- If InterestRate >= 5%, increase by 0.5%');

DBMS\_OUTPUT.PUT\_LINE('--------------------------------------------------');

-- Start processing each loan

OPEN cur\_loans;

LOOP

FETCH cur\_loans INTO rec\_loan;

EXIT WHEN cur\_loans%NOTFOUND;

-- Print current loan interest

DBMS\_OUTPUT.PUT\_LINE('Loan ID : ' || rec\_loan.LoanID);

DBMS\_OUTPUT.PUT\_LINE('Old InterestRate: ' || rec\_loan.InterestRate || '%');

-- Apply policy

IF rec\_loan.InterestRate < 5 THEN

v\_new\_rate := rec\_loan.InterestRate + 1;

ELSE

v\_new\_rate := rec\_loan.InterestRate + 0.5;

END IF;

-- Update new interest rate in the table

UPDATE Loans

SET InterestRate = v\_new\_rate

WHERE LoanID = rec\_loan.LoanID;

-- Print result

DBMS\_OUTPUT.PUT\_LINE('New InterestRate: ' || v\_new\_rate || '%');

DBMS\_OUTPUT.PUT\_LINE('--------------------------------------------------');

END LOOP;

CLOSE cur\_loans;

COMMIT;

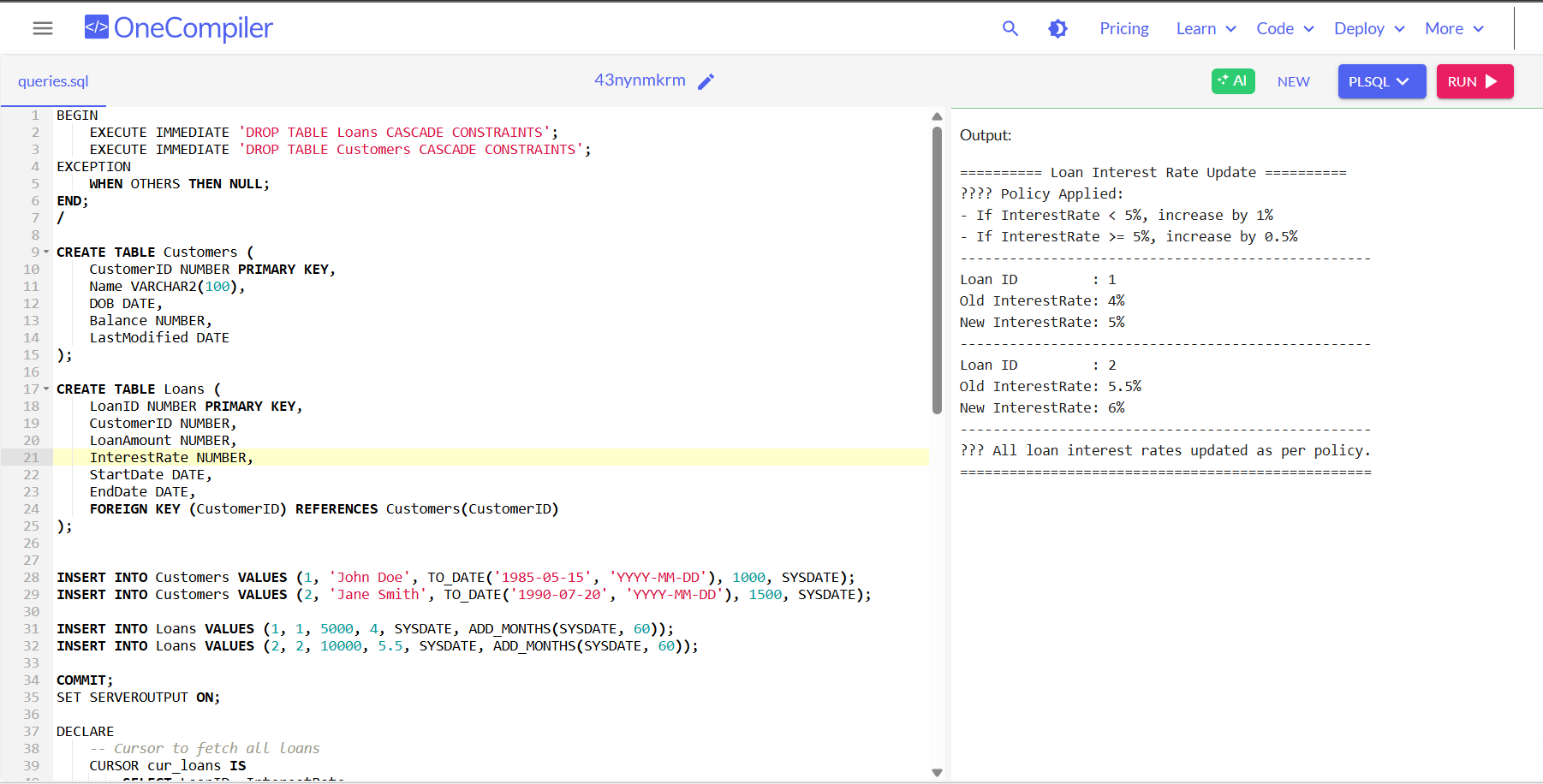
DBMS\_OUTPUT.PUT\_LINE(' All loan interest rates updated as per policy.');

DBMS\_OUTPUT.PUT\_LINE('==================================================');

END;

/

**Output:**

****

**Exercise 7: Packages**

**Scenario 1:** Group all customer-related procedures and functions into a package.

* + **Question:** Create a package CustomerManagement with procedures for adding a new customer, updating customer details, and a function to get customer balance.

**Solution:**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer (

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_DOB IN DATE,

p\_Balance IN NUMBER

);

PROCEDURE UpdateCustomerDetails (

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_DOB IN DATE,

p\_Balance IN NUMBER

);

FUNCTION GetCustomerBalance (

p\_CustomerID IN NUMBER

) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer (

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_DOB IN DATE,

p\_Balance IN NUMBER

) IS

BEGIN

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Added Customer: ' || p\_Name || ' (ID: ' || p\_CustomerID || ') with Balance: ' || p\_Balance);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_CustomerID || ' already exists.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Unexpected error in AddCustomer: ' || SQLERRM);

END AddCustomer;

PROCEDURE UpdateCustomerDetails (

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_DOB IN DATE,

p\_Balance IN NUMBER

) IS

v\_exists NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_exists FROM Customers WHERE CustomerID = p\_CustomerID;

IF v\_exists = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Cannot update: Customer with ID ' || p\_CustomerID || ' does not exist.');

RETURN;

END IF;

UPDATE Customers

SET Name = p\_Name,

DOB = p\_DOB,

Balance = p\_Balance,

LastModified = SYSDATE

WHERE CustomerID = p\_CustomerID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(' Updated Customer ID: ' || p\_CustomerID || ' to Name: ' || p\_Name || ', Balance: ' || p\_Balance);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Unexpected error in UpdateCustomerDetails: ' || SQLERRM);

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance (

p\_CustomerID IN NUMBER

) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

SELECT Balance

INTO v\_Balance

FROM Customers

WHERE CustomerID = p\_CustomerID;

DBMS\_OUTPUT.PUT\_LINE(' Balance retrieval successful for Customer ID ' || p\_CustomerID || ': ' || v\_Balance);

RETURN v\_Balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No customer found with ID ' || p\_CustomerID);

RETURN NULL;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Unexpected error in GetCustomerBalance: ' || SQLERRM);

RETURN NULL;

END GetCustomerBalance;

END CustomerManagement;

/

SET SERVEROUTPUT ON SIZE 1000000;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('============================================');

DBMS\_OUTPUT.PUT\_LINE(' CUSTOMER MANAGEMENT PACKAGE TEST STARTED');

DBMS\_OUTPUT.PUT\_LINE('============================================');

DBMS\_OUTPUT.PUT\_LINE(CHR(10)); -- Blank line

-- Add customers

DBMS\_OUTPUT.PUT\_LINE('--- Adding Customers ---');

CustomerManagement.AddCustomer(10, 'Samuel Green', TO\_DATE('1988-11-23', 'YYYY-MM-DD'), 3000);

CustomerManagement.AddCustomer(11, 'Linda Blue', TO\_DATE('1992-04-10', 'YYYY-MM-DD'), 4500);

CustomerManagement.AddCustomer(10, 'Duplicate Sam', TO\_DATE('1980-01-01', 'YYYY-MM-DD'), 1000); -- Duplicate ID test

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || CHR(10) || '--------------------------------------------' || CHR(10));

-- Display balances after addition

DBMS\_OUTPUT.PUT\_LINE('--- Balances After Addition ---');

DBMS\_OUTPUT.PUT\_LINE('Customer 10 balance: ' || NVL(TO\_CHAR(CustomerManagement.GetCustomerBalance(10)), 'N/A'));

DBMS\_OUTPUT.PUT\_LINE('Customer 11 balance: ' || NVL(TO\_CHAR(CustomerManagement.GetCustomerBalance(11)), 'N/A'));

DBMS\_OUTPUT.PUT\_LINE('Customer 99 balance (nonexistent): ' || NVL(TO\_CHAR(CustomerManagement.GetCustomerBalance(99)), 'N/A'));

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || CHR(10) || '--------------------------------------------' || CHR(10));

-- Update customer details

DBMS\_OUTPUT.PUT\_LINE('--- Updating Customers ---');

CustomerManagement.UpdateCustomerDetails(10, 'Samuel G. Updated', TO\_DATE('1988-11-23', 'YYYY-MM-DD'), 3500);

CustomerManagement.UpdateCustomerDetails(99, 'Ghost Customer', TO\_DATE('1970-01-01', 'YYYY-MM-DD'), 0); -- Non-existent

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || CHR(10) || '--------------------------------------------' || CHR(10));

-- Display balances after update

DBMS\_OUTPUT.PUT\_LINE('--- Balances After Update ---');

DBMS\_OUTPUT.PUT\_LINE('Customer 10 balance: ' || NVL(TO\_CHAR(CustomerManagement.GetCustomerBalance(10)), 'N/A'));

DBMS\_OUTPUT.PUT\_LINE('Customer 11 balance: ' || NVL(TO\_CHAR(CustomerManagement.GetCustomerBalance(11)), 'N/A'));

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || CHR(10));

DBMS\_OUTPUT.PUT\_LINE('============================================');

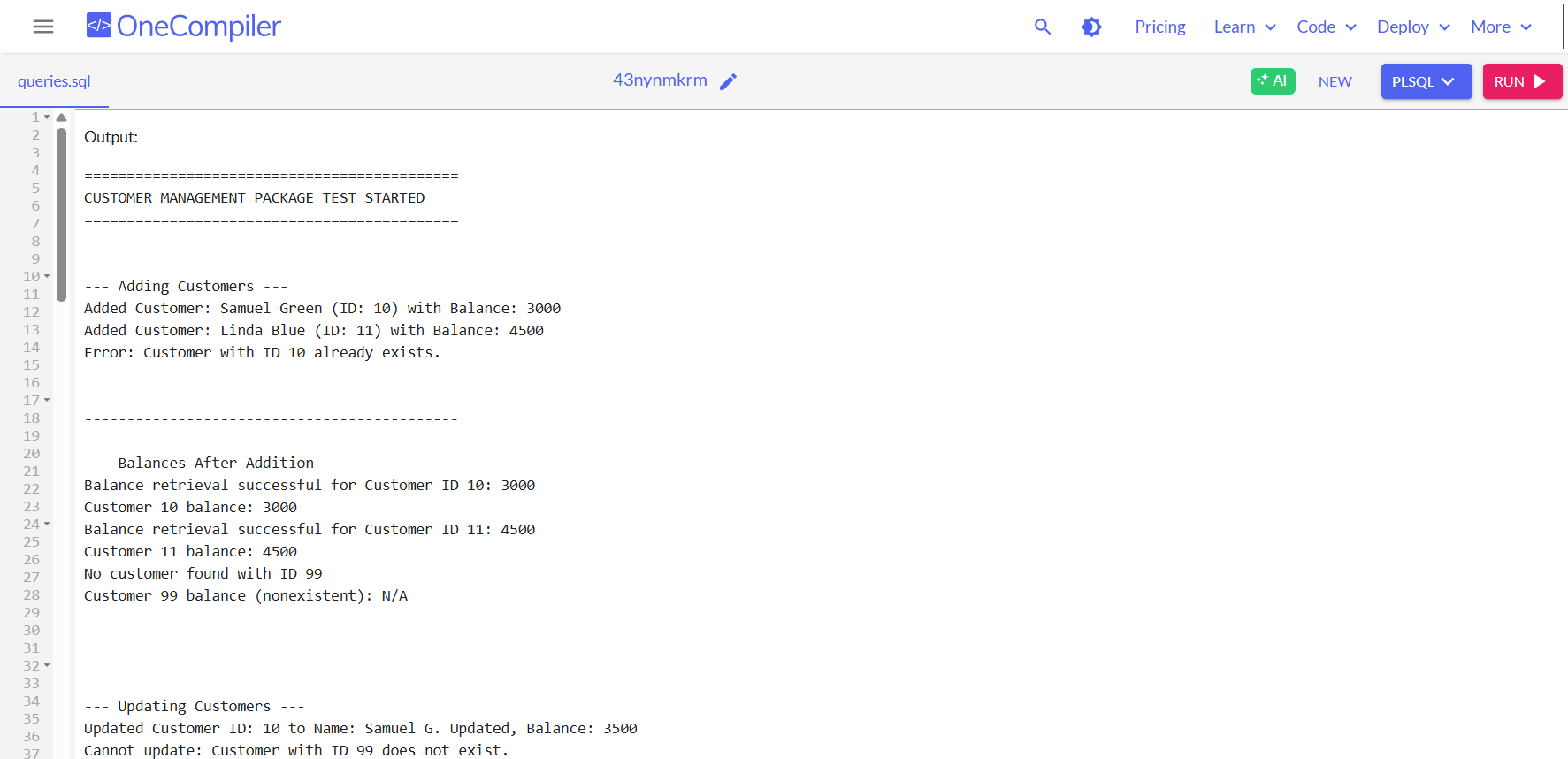
DBMS\_OUTPUT.PUT\_LINE(' CUSTOMER MANAGEMENT PACKAGE TEST COMPLETED');

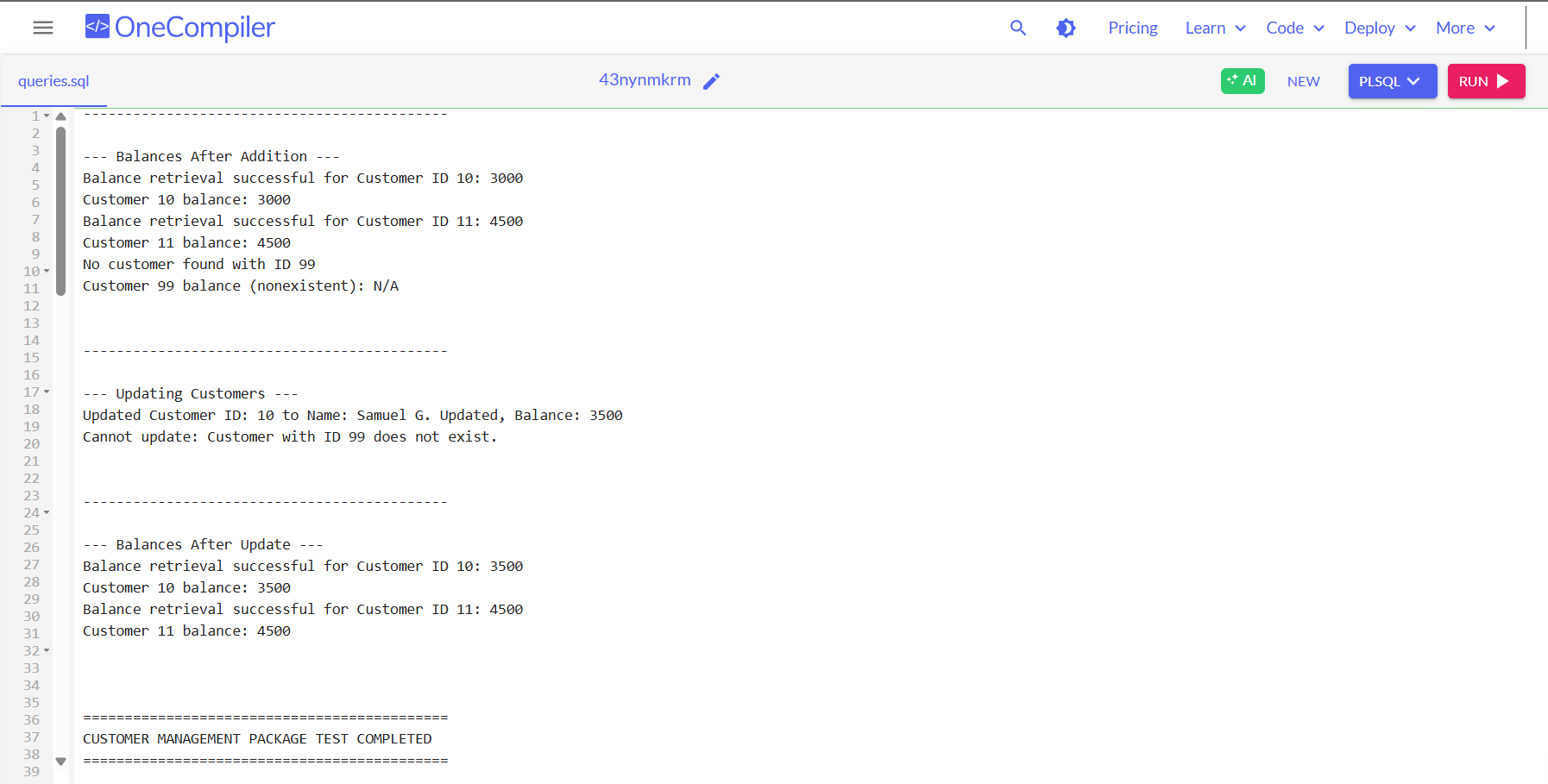
DBMS\_OUTPUT.PUT\_LINE('============================================');

END;

/

**Output:**

****

****

**Scenario 2:** Create a package to manage employee data.

* + **Question:** Write a package EmployeeManagement with procedures to hire new employees, update employee details, and a function to calculate annual salary.

**Solution:**

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER, -- Monthly salary

Department VARCHAR2(50),

HireDate DATE

);

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee (

p\_EmployeeID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2,

p\_HireDate IN DATE

);

PROCEDURE UpdateEmployee (

p\_EmployeeID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2

);

FUNCTION CalculateAnnualSalary (

p\_EmployeeID IN NUMBER

) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee (

p\_EmployeeID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2,

p\_HireDate IN DATE

) IS

BEGIN

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (p\_EmployeeID, p\_Name, p\_Position, p\_Salary, p\_Department, p\_HireDate);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Hired: ' || p\_Name || ' (ID: ' || p\_EmployeeID || ')');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID ' || p\_EmployeeID || ' already exists.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error in HireEmployee: ' || SQLERRM);

END;

PROCEDURE UpdateEmployee (

p\_EmployeeID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2

) IS

BEGIN

UPDATE Employees

SET Name = p\_Name,

Position = p\_Position,

Salary = p\_Salary,

Department = p\_Department

WHERE EmployeeID = p\_EmployeeID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Employee ID ' || p\_EmployeeID || ' not found.');

ELSE

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(' Updated: ' || p\_Name || ' (ID: ' || p\_EmployeeID || ')');

END IF;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error in UpdateEmployee: ' || SQLERRM);

END;

FUNCTION CalculateAnnualSalary (

p\_EmployeeID IN NUMBER

) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_EmployeeID;

RETURN v\_salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Employee ID ' || p\_EmployeeID || ' not found.');

RETURN NULL;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error in CalculateAnnualSalary: ' || SQLERRM);

RETURN NULL;

END;

END EmployeeManagement;

/

SET SERVEROUTPUT ON;

BEGIN

-- Hire Employees

DBMS\_OUTPUT.PUT\_LINE('--- Hiring Employees ---');

EmployeeManagement.HireEmployee(1001, 'Alice Johnson', 'Manager', 7000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

EmployeeManagement.HireEmployee(1002, 'Bob Brown', 'Developer', 6000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

EmployeeManagement.HireEmployee(1001, 'Duplicate Alice', 'Tester', 5000, 'QA', SYSDATE); -- Duplicate ID test

DBMS\_OUTPUT.PUT\_LINE(CHR(10));

-- Update Employees

DBMS\_OUTPUT.PUT\_LINE('--- Updating Employee ---');

EmployeeManagement.UpdateEmployee(1002, 'Bob B.', 'Senior Developer', 6500, 'IT');

EmployeeManagement.UpdateEmployee(9999, 'Ghost', 'None', 0, 'Nowhere'); -- Non-existent

DBMS\_OUTPUT.PUT\_LINE(CHR(10));

-- Calculate Annual Salary

DBMS\_OUTPUT.PUT\_LINE('--- Calculating Annual Salaries ---');

DBMS\_OUTPUT.PUT\_LINE('Annual Salary for Employee 1001: ' || NVL(TO\_CHAR(EmployeeManagement.CalculateAnnualSalary(1001)), 'N/A'));

DBMS\_OUTPUT.PUT\_LINE('Annual Salary for Employee 1002: ' || NVL(TO\_CHAR(EmployeeManagement.CalculateAnnualSalary(1002)), 'N/A'));

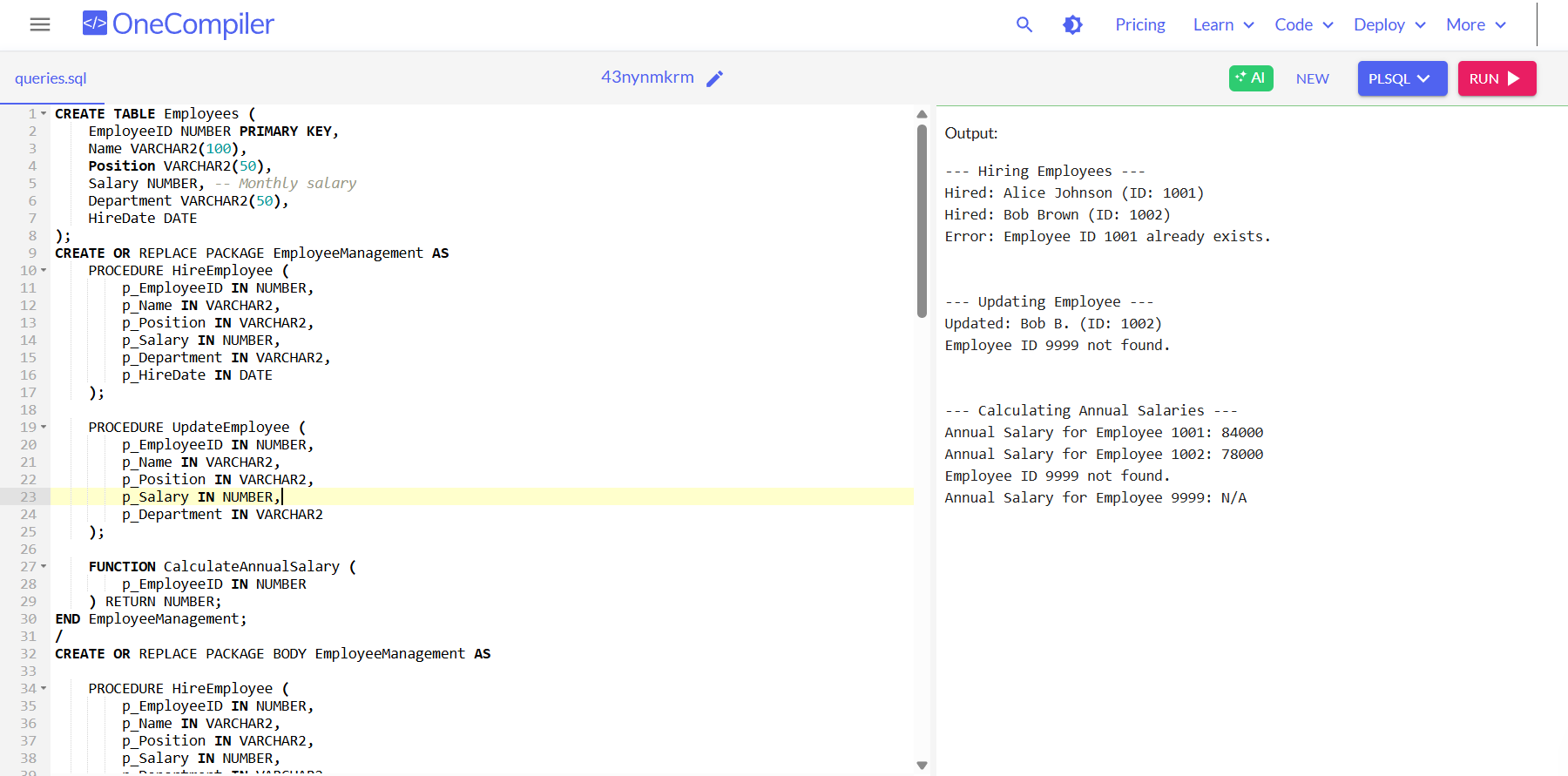
DBMS\_OUTPUT.PUT\_LINE('Annual Salary for Employee 9999: ' || NVL(TO\_CHAR(EmployeeManagement.CalculateAnnualSalary(9999)), 'N/A'));

DBMS\_OUTPUT.PUT\_LINE('');

END;

/

**Output:**



**Scenario 3:** Group all account-related operations into a package.

* + **Question:** Create a package AccountOperations with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

**Solution:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS';

EXCEPTION

WHEN OTHERS THEN

IF SQLCODE != -942 THEN RAISE; END IF;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';

EXCEPTION

WHEN OTHERS THEN

IF SQLCODE != -942 THEN RAISE; END IF;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

COMMIT;

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount (

p\_AccountID IN NUMBER,

p\_CustomerID IN NUMBER,

p\_AccountType IN VARCHAR2,

p\_InitialBalance IN NUMBER

);

PROCEDURE CloseAccount (

p\_AccountID IN NUMBER

);

FUNCTION GetTotalBalance (

p\_CustomerID IN NUMBER

) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount (

p\_AccountID IN NUMBER,

p\_CustomerID IN NUMBER,

p\_AccountType IN VARCHAR2,

p\_InitialBalance IN NUMBER

) IS

BEGIN

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (p\_AccountID, p\_CustomerID, p\_AccountType, p\_InitialBalance, SYSDATE);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Account ' || p\_AccountID || ' opened for Customer ' || p\_CustomerID);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account ID ' || p\_AccountID || ' already exists.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error in OpenAccount: ' || SQLERRM);

END;

PROCEDURE CloseAccount (

p\_AccountID IN NUMBER

) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_AccountID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('No account found with ID ' || p\_AccountID);

ELSE

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Account ' || p\_AccountID || ' has been closed.');

END IF;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error in CloseAccount: ' || SQLERRM);

END;

FUNCTION GetTotalBalance (

p\_CustomerID IN NUMBER

) RETURN NUMBER IS

v\_total\_balance NUMBER := 0;

BEGIN

SELECT NVL(SUM(Balance), 0)

INTO v\_total\_balance

FROM Accounts

WHERE CustomerID = p\_CustomerID;

RETURN v\_total\_balance;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error in GetTotalBalance: ' || SQLERRM);

RETURN NULL;

END;

END AccountOperations;

/

SET SERVEROUTPUT ON;

BEGIN

-- Open accounts

DBMS\_OUTPUT.PUT\_LINE('--- Opening Accounts ---');

AccountOperations.OpenAccount(101, 1, 'Savings', 2000);

AccountOperations.OpenAccount(102, 1, 'Checking', 1500);

AccountOperations.OpenAccount(103, 2, 'Savings', 1000);

AccountOperations.OpenAccount(101, 1, 'Savings', 999); -- Duplicate

DBMS\_OUTPUT.PUT\_LINE('');

DBMS\_OUTPUT.PUT\_LINE('--------------------------------------------');

DBMS\_OUTPUT.PUT\_LINE('');

-- Get total balances

DBMS\_OUTPUT.PUT\_LINE('--- Customer Balances ---');

DBMS\_OUTPUT.PUT\_LINE('Total balance for Customer 1: ' || AccountOperations.GetTotalBalance(1));

DBMS\_OUTPUT.PUT\_LINE('Total balance for Customer 2: ' || AccountOperations.GetTotalBalance(2));

DBMS\_OUTPUT.PUT\_LINE('Total balance for Customer 99: ' || NVL(TO\_CHAR(AccountOperations.GetTotalBalance(99)), 'N/A'));

DBMS\_OUTPUT.PUT\_LINE('');

DBMS\_OUTPUT.PUT\_LINE('--------------------------------------------');

DBMS\_OUTPUT.PUT\_LINE('');

-- Close accounts

DBMS\_OUTPUT.PUT\_LINE('--- Closing Accounts ---');

AccountOperations.CloseAccount(102);

AccountOperations.CloseAccount(999); -- Not found

DBMS\_OUTPUT.PUT\_LINE('');

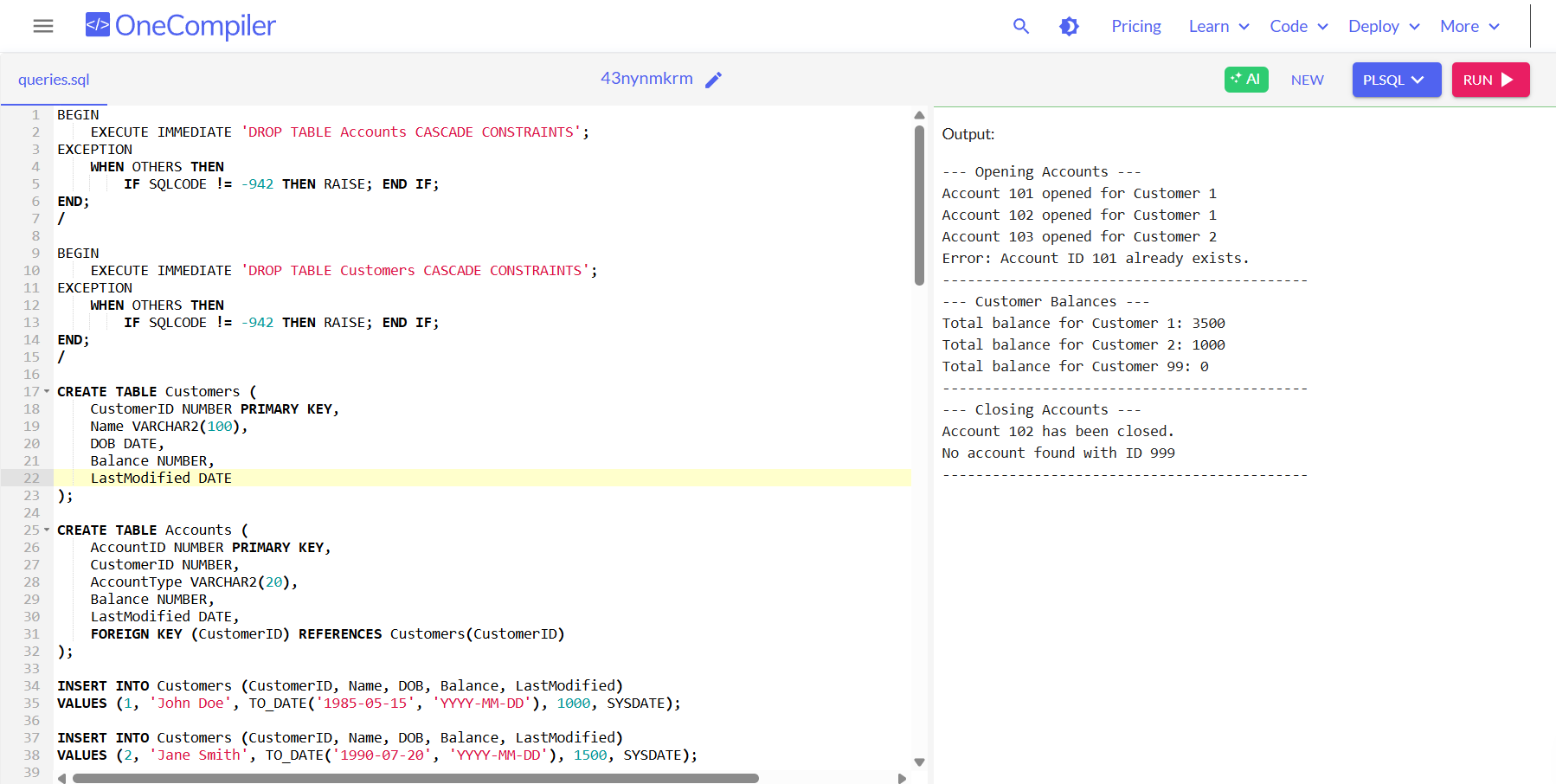
DBMS\_OUTPUT.PUT\_LINE('--------------------------------------------');

DBMS\_OUTPUT.PUT\_LINE('');

END;

/

**Output:**

****